Ant User Manual

by

- Stephane Bailliez (sbailliez@imediation.com)
- Jacques Bergeron (jacques.bergeron@dogico.com)
- Stefan Bodewig (stefan.bodewig@epost.de)
- Patrick Chanezon (chanezon@netscape.com)
- James Duncan Davison (duncan@x180.com)
- Tom Dimock (tad1@cornell.edu)
- Diane Holt (holtdl@yahoo.com)
- Bill Kelly (bill.kelly@softwired-inc.com)
- Arnout J. Kuiper (ajkuiper@wxs.nl)
- Conor MacNeill (conor@cortexebusiness.com.au)
- Stefano Mazzocchi (stefano@apache.org)
- Erik Meade (emeade@geekfarm.org)
- Sam Ruby (rubys@us.ibm.com)
- Nico Seessle (nico@seessle.de)
- Jon S. Stevens (jon@latchkey.com)
- Roger Vaughn (rvaughn@seaconinc.com)
- Dave Walend (dwalend@cs.tufts.edu)
- Phillip Wells (philwells@rocketmail.com)

Version: @VERSION@
$Id: credits.html,v 1.1.2.2 2001/02/15 11:26:13 conor Exp $

Copyright © 2000,2001 Apache Software Foundation. All rights Reserved.

Table of Contents

1 Introduction 1
2 Installing Ant 2
2.1 Getting Ant 2
2.2 System Requirements 2
2.3 Installing Ant 2
2.4 Building Ant 3
2.5 Running Ant 5
2.6 Library Dependencies 6
3 Using Ant 8
3.1 Writing a Simple Buildfile 8
4 Concepts and Types 16
4.1 Directory-based Tasks 16
4.2 PatternSet 19
4.3 FileSet 20
4.4 Mapping File Names 21
5 Core Tasks 25
5.1 Ant 25
5.2 AntCall 26
5.3 AntStructure 27
5.4 Apply 28
5.5 Available 30
5.6 Chmod 32
5.7 Copy 34
5.8 Copydir 36
5.9 Copyfile 38
5.10 Cvs 39
5.11 Delete 41
5.12 Deltree 43
5.13 Echo 44
5.14 Exec 45
5.15 ExecOn 47
5.16 Fail 49
5.17 Filter 50
5.18 FixCRLF 51
5.19 GenKey 54
5.20 Get 55
5.21 GUnzip 56
5.22 GZip 57
5.23 Jar 58
5.24 Java 61
5.25 Javac 63
5.26 Javadoc/Javadoc2 67
5.27 Mail 72
5.28 Mkdir 73

5.29 Move 74
5.30 Patch 76
5.31 Property 77
5.32 Rename 80
5.33 Replace 81
5.34 Rmic 84
5.35 SignJar 87
5.36 Sql 88
5.37 Style 92
5.38 Tar 95
5.39 Taskdef 98
5.40 Touch 99
5.41 Tstamp 100
5.42 Unjar/Unwar/Unzip 101
5.43 Untar 102
5.44 Unjar/Unwar/Unzip 103
5.45 Unjar/Unwar/Unzip 104
5.46 Uptodate 105
5.47 War 106
5.48 Zip 108
6 Optional Tasks 111
6.1 .NET classes for Ant 111
6.2 ANTLR 117
6.3 Cab 118
6.4 ClearCase Support 120
6.5 Depend 123
6.6 Ant EJB Tasks User Manual 125
6.7 FTP 138
6.8 JavaCC 142
6.9 Javah 145
6.10 JJTree 147
6.11 Jlink 149
6.12 JUnit 151
6.13 JUnitReport 155
6.14 MParse 158
6.15 Native2Ascii 159
6.16 NetRexxC 161
6.17 Perforce Tasks User Manual 164
6.18 Ant PropertyFile Task User Manual 169
6.19 RenameExtensions 172
6.20 Script 173
6.21 Sound 175
6.22 Stylebook 177
6.23 Telnet 178
6.24 Test 180
6.25 VssGet 181
6.26 VssLabel 182

7 IDE Integration 183
7.1 About Antidote 183
7.2 AntRunner For JBuilder (unbundled) 185
7.3 NetBeans Forte for Java (unbundled) 185
7.4 Visual Age for Java Tasks and Plugin User Manual 185
8 Developing with Ant 195
8.1 Writing Your Own Task 195
8.2 Build Events 197
9 Ant Resources 198
9.1 FAQ 198
10 License 199
11 Feedback 200

1 Introduction

Ant is a Java-based build tool. In theory, it is kind of like *make*, without *make*'s wrinkles.

1.0.1 Why?

Why another build tool when there is already *make*, *gnumake*, *nmake*, *jam*, and others? Because all those tools have limitations that Ant's original author couldn't live with when developing software across multiple platforms. Make-like tools are inherently shell-based: they evaluate a set of dependencies, then execute commands not unlike what you would issue on a shell. This means that you can easily extend these tools by using or writing any program for the OS that you are working on; however, this also means that you limit yourself to the OS, or at least the OS type, such as Unix, that you are working on.

Makefiles are inherently evil as well. Anybody who has worked on them for any time has run into the dreaded tab problem. "Is my command not executing because I have a space in front of my tab?!!" said the original author of Ant way too many times. Tools like Jam took care of this to a great degree, but still have yet another format to use and remember.

Ant is different. Instead of a model where it is extended with shell-based commands, Ant is extended using Java classes. Instead of writing shell commands, the configuration files are XML-based, calling out a target tree where various tasks get executed. Each task is run by an object that implements a particular Task interface.

Granted, this removes some of the expressive power that is inherent in being able to construct a shell command such as `'find . -name foo -exec rm {}'`, but it gives you the ability to be cross-platform – to work anywhere and everywhere. And hey, if you really need to execute a shell command, Ant has an <exec> task that allows different commands to be executed based on the OS it is executing on.

2 Installing Ant

2.1 Getting Ant

2.1.1 Binary edition

The latest stable version of Ant can be downloaded from http://jakarta.apache.org/builds/ant/release/v1.3/bin/. If you like living on the edge, you can download the latest version from http://jakarta.apache.org/builds/ant/nightly/.

2.1.2 Source Edition

If you prefer the source edition, you can download the source for the latest Ant release from http://jakarta.apache.org/builds/ant/release/v1.3/src/. Again, if you prefer the edge, you can access the code as it is being developed via CVS. The Jakarta website has details on accessing CVS (http://jakarta.apache.org/site/cvsindex.html). Please checkout the jakarta-ant module. See the section Building Ant [p 3] on how to build Ant from the source code. You can also access the Ant CVS repository (http://jakarta.apache.org/cvsweb/index.cgi/jakarta-ant/) on-line.

2.2 System Requirements

To build and use Ant, you must have a JAXP-compliant XML parser installed and available on your class-path.

Both the binary and source distributions of Ant include the reference implementation of JAXP 1.0. Please see http://java.sun.com/xml/ for more information. If you wish to use a different JAXP-compliant parser, you should remove `jaxp.jar` and `parser.jar` from Ant's `lib` directory. You can then either put the jars from your preferred parser into Ant's `lib` directory or put the jars on the system classpath.

For the current version of Ant, you will also need a JDK installed on your system, version 1.1 or later. A future version of Ant will require JDK 1.2 or later.

2.3 Installing Ant

The binary distribution of Ant consists of three directories: `bin`, `docs` and `lib` Only the `bin` and `lib` directories are required to run Ant. To install Ant, choose a directory and copy the distribution file there. This directory will be known as ANT_HOME. Before you can run ant there is some additional set up you will need to do:

- Add the `bin` directory to your path.
- Set the `ANT_HOME` environment variable to the directory where you installed Ant.
- Optionally, set the `JAVA_HOME` environment variable (see the Advanced [p 3] section below). This should be set to the directory where your JDK is installed.

- If you are using Ant's optional tasks then you may need to add additional jars to your `lib` directory or to your CLASSPATH. See Library Dependencies [p 6] for for a list of jar requirements for various optional tasks.

Note: Do not install Ant's ant.jar file into the lib/ext directory of the JDK/JRE. Ant is an application, whilst the extension directory is intended for JDK extensions. In particular there are security restrictions on the classes which may be loaded by an extension.

2.3.1 Windows

Assume Ant is installed in `c:\ant\`. The following sets up the environment:

```
set ANT_HOME=c:\ant
set JAVA_HOME=c:\jdk1.2.2
set PATH=%PATH%;%ANT_HOME%\bin
```

2.3.2 Unix (bash)

Assume Ant is installed in `/usr/local/ant`. The following sets up the environment:

```
export ANT_HOME=/usr/local/ant
export JAVA_HOME=/usr/local/jdk-1.2.2
export PATH=${PATH}:${ANT_HOME}/bin
```

2.3.3 Advanced

There are lots of variants that can be used to run Ant. What you need is at least the following:

- The classpath for Ant must contain `ant.jar` and any jars/classes needed for your chosen JAXP-compliant XML parser.
- When you need JDK functionality (such as for the javac [p 63] task or the rmic [p 84] task), then for JDK 1.1, the `classes.zip` file of the JDK must be added to the classpath; for JDK 1.2 or JDK 1.3, `tools.jar` must be added. The scripts supplied with Ant, in the `bin` directory, will add the required JDK classes automatically, if the `JAVA_HOME` environment variable is set.
- When you are executing platform-specific applications, such as the exec [p 45] task or the cvs [p 39] task, the property `ant.home` must be set to the directory containing where you installed Ant. Again this is set by the Ant scripts to the value of the ANT_HOME environment variable.

2.4 Building Ant

To build Ant from source, you can either install the Ant source distribution or checkout the jakarta-ant module from CVS.

Once you have installed the source, change into the installation directory.

Set the JAVA_HOME environment variable to the directory where the JDK is installed. See Installing Ant [p 2] for examples on how to do this for your operating system.

Make sure you have downloaded any auxiliary jars required to build tasks you are interested in. These should either be available on the CLASSPATH or added to the lib/optional directory. See Library Dependencies [p 6] for for a list of jar requirements for various features. Note that this will make the auxiliary jars available for the building of Ant only. For running Ant you will still need to make the jars available as described under Installing Ant [p 2] .

Your are now ready to build Ant:

```
build -Ddist.dir=<directory_to_contain_Ant_distribution>
dist  (Windows)

build.sh -Ddist.dir=<directory_to_contain_Ant_distribution>
dist  (Unix)
```

This will create a binary distribution of Ant in the directory you specified.

The above action does the following:

- If necessary it will bootstrap the Ant code. Bootstrapping involves the manual compilation of enough Ant code to be able to run Ant. The bootstrapped Ant is used for the remainder of the build steps.
- Invokes the bootstrapped Ant with the parameters passed to the build script. In this case, these parameters define an Ant property value and specify the "dist" target in Ant's own build.xml file.

On most occasions you will not need to explicitly bootstrap Ant since the build scripts do that for you. If however, the build file you are using makes use of features not yet compiled into the bootstrapped Ant, you will need to manually bootstrap. Run bootstrap.bat (Windows) or bootstrap.sh (UNIX) to build a new bootstrap version of Ant.

If you wish to install the build into the current ANT_HOME directory, you can use:

```
build install  (Windows)

build.sh install  (Unix)
```

You can avoid the lengthy Javadoc step, if desired, with:

```
build install-lite  (Windows)

build.sh install-lite  (Unix)
```

This will only install the bin and lib directories.

Both the install and install-lite targets will overwrite the current Ant version in ANT_HOME.

2.5 Running Ant

Running Ant is simple, when you installed it as described in the previous section. Just type `ant`.

When nothing is specified, Ant looks for a `build.xml` file in the current directory. If found, it uses that file as the buildfile. If you use the `-find` option, Ant will search for a buildfile in the parent directory, and so on, until the root of the filesystem has been reached. To make Ant use another buildfile, use the command-line option `-buildfile` *`file`*, where *file* is the buildfile you want to use.

You can also set properties that override properties specified in the buildfile (see the property [p 77] task). This can be done with the `-D`*`property=value`* option, where *property* is the name of the property, and *value* is the value for that property. This can also be used to pass in the value of some environment variables. You can also access environment variables using the property [p 77] task. Just pass `-DMYVAR=%MYVAR%` (Windows) or `-DMYVAR=$MYVAR` (Unix) to Ant – you can then access these variables inside your buildfile as `${MYVAR}`.

Two more options are: `-quiet`, which instructs Ant to print less information on the console when running, and `-verbose`, which causes Ant to print additional information to the console.

It is also possible to specify one or more targets that should be executed. When omitted, the target that is specified in the `default` attribute of the `<project>` tag is used.

The `-projecthelp` option gives a list of this project's targets. First those with a description, then those without one.

Command-line option summary:

```
ant [options] [target [target2 [target3] ...]]
Options:
-help                  print this message
-projecthelp           print project help information
-version               print the version information and exit
-quiet                 be extra quiet
-verbose               be extra verbose
-debug                 print debugging information
-emacs                 produce logging information without adornments
-logfile file          use given file for log output
-logger classname      the class that is to perform logging
-listener classname    add an instance of class as a project listener
-buildfile file        use specified buildfile
-find file             search for buildfile towards the root of the filesystem and use the first one found
-Dproperty=value       set property to value
```

2.5.1 Examples

```
ant
```

runs Ant using the `build.xml` file in the current directory, on the default target.

```
ant -buildfile test.xml
```

runs Ant using the `test.xml` file in the current directory, on the default target.

```
ant -buildfile test.xml dist
```

runs Ant using the `test.xml` file in the current directory, on a target called `dist`.

```
ant -buildfile test.xml -Dbuild=build/classes dist
```

runs Ant using the `test.xml` file in the current directory, on a target called `dist`, setting the `build` property to the value `build/classes`.

2.5.2 Running Ant by Hand

If you have installed Ant in the do-it-yourself way, Ant can be started with:

```
java -Dant.home=c:\ant org.apache.tools.ant.Main [options] [target]
```

These instructions actually do exactly the same as the `ant` command. The options and target are the same as when running Ant with the `ant` command. This example assumes you have set your classpath to include:

- `ant.jar`
- jars/classes for your XML parser
- the JDK's required jar/zip files

2.6 Library Dependencies

The following libraries are needed in your CLASSPATH or in the install directory's `lib` directory if you are using the indicated feature. Note that only one of the regexp libraries is needed for use with the mappers.

Jar Name	Needed For	Available At
jakarta-regexp-1.2.jar	regexp type with mappers	jakarta.apache.org/regexp/
jakarta-oro-2.0.1.jar	regexp type with mappers and the perforce tasks	jakarta.apache.org/oro/
junit.jar	junit tasks	www.junit.org
stylebook.jar	stylebook task	CVS repository of xml.apache.org
testlet.jar	test task	java.apache.org/framework
antlr.jar	antlr task	www.antlr.org
bsf.jar	script task	oss.software.ibm.com/developerworks/projects/bsf
netrexx.jar	netrexx task	www2.hursley.ibm.com/netrexx
rhino.jar	javascript with script task	www.mozilla.org
jpython.jar	python with script task	www.jpython.org
netcomponents.jar	ftp and telnet tasks	www.savarese.org/oro/downloads

3 Using Ant

3.1 Writing a Simple Buildfile

Ant's buildfiles are written in XML. Each buildfile contains one project.

Each task element of the buildfile can have an `id` attribute and can later be referred to by the value supplied to this. The value has to be unique. (For additional information, see the Tasks [p 9] section below.)

3.1.1 Projects

A *project* has three attributes:

Attribute	Description	Required
name	the name of the project.	No
default	the default target to use when no target is supplied.	Yes
basedir	the base directory from which all path calculations are done. This attribute might be overridden by setting the "basedir" property beforehand. When this is done, it must be omitted in the project tag. If neither the attribute nor the property have been set, the parent directory of the buildfile will be used.	No

Each project defines one or more *targets*. A target is a set of *tasks* you want to be executed. When starting Ant, you can select which target(s) you want to have executed. When no target is given, the project's default is used.

3.1.2 Targets

A target can depend on other targets. You might have a target for compiling, for example, and a target for creating a distributable. You can only build a distributable when you have compiled first, so the distribute target *depends on* the compile target. Ant resolves these dependencies.

It should be noted, however, that Ant's `depends` attribute only specifies the *order* in which targets should be executed – it does not affect whether the target that specifies the dependency(s) gets executed if the dependent target(s) did not (need to) run.

Ant tries to execute the targets in the `depends` attribute in the order they appear (from left to right). Keep in mind that it is possible that a target can get executed earlier when an earlier target depends on it:

```
<target name="A"/>
<target name="B" depends="A"/>
<target name="C" depends="B"/>
<target name="D" depends="C,B,A"/>
```

Suppose we want to execute target D. From its `depends` attribute, you might think that first target C, then B and then A is executed. Wrong! C depends on B, and B depends on A, so first A is executed, then B, then C, and finally D.

A target gets executed only once, even when more than one target depends on it (see the previous example).

A target also has the ability to perform its execution if (or unless) a property has been set. This allows, for example, better control on the building process depending on the state of the system (java version, OS, command-line property defines, etc.). To make a target *sense* this property, you should add the `if` (or `unless`) attribute with the name of the property that the target should react to. For example:

```
<target name="build-module-A" if="module-A-present"/>

<target name="build-own-fake-module-A" unless="module-A-present"/>
```

If no `if` and no `unless` attribute is present, the target will always be executed.

The optional `description` attribute can be used to provide a one-line description of this target, which is printed by the `-projecthelp` command-line option.

It is a good practice to place your tstamp [p 100] tasks in a so-called *initialization* target, on which all other targets depend. Make sure that target is always the first one in the depends list of the other targets. In this manual, most initialization targets have the name "init".

A target has the following attributes:

Attribute	Description	Required
name	the name of the target.	Yes
depends	a comma-separated list of names of targets on which this target depends.	No
if	the name of the property that must be set in order for this target to execute.	No
unless	the name of the property that must not be set in order for this target to execute.	No
description	a short description of this target's function.	No

3.1.3 Tasks

A task is a piece of code that can be executed.

A task can have multiple attributes (or arguments, if you prefer). The value of an attribute might contain references to a property. These references will be resolved before the task is executed.

Tasks have a common structure:

```
<name attribute1="value1" attribute2="value2" ... />
```

where *name* is the name of the task, *attributeN* is the attribute name, and *valueN* is the value for this attribute.

There is a set of built-in tasks [p 16] , along with a number of optional tasks [p 111] , but it is also very easy to write your own [p 195] .

All tasks share a task name attribute. The value of this attribute will be used in the logging messages generated by Ant.

Tasks can be assigned an `id` attribute:

```
<taskname id="taskID" ... />
```

where *taskname* is the name of the task, and *taskID* is a unique name for this task. You can refer to the corresponding task object in scripts or other tasks via this name. For example, in scripts you could do:

```
<script ... >
  task1.setFoo("bar");
</script>
```

to set the `foo` attribute of this particular task instance. In another task (written in Java), you can access the instance via `project.getReference("task1")`.

Note[1]: If "task1" has not been run yet, then it has not been configured (ie., no attributes have been set), and if it is going to be configured later, anything you've done to the instance may be overwritten.

Note[2]: Future versions of Ant will most likely *not* be backward-compatible with this behaviour, since there will likely be no task instances at all, only proxies.

3.1.4 Properties

A project can have a set of properties. These might be set in the buildfile by the property task [p 77] , or might be set outside Ant. A property has a name and a value. Properties may be used in the value of task attributes. This is done by placing the property name between "`${`" and "`}`" in the attribute value. For example, if there is a "builddir" property with the value "build", then this could be used in an attribute like this: `${builddir}/classes`. This is resolved as `build/classes`.

3.1.5 Built-in Properties

Ant provides access to all system properties as if they had been defined using a `<property>` task. For example, `${os.name}` expands to the name of the operating system.

In addition, Ant has some built-in properties:

```
basedir             the absolute path of the project's basedir (as set
                    with the basedir attribute of <project>).
ant.file            the absolute path of the buildfile.
ant.project.name    the name of the project that is currently executing;
                    it is set in the name attribute of <project>.
ant.java.version    the JVM version Ant detected; currently it can hold
                    the values "1.1", "1.2" and "1.3".
```

3.1.6 Example

```
<project name="MyProject" default="dist" basedir=".">

  <!-- set global properties for this build -->
  <property name="src" value="."/>
  <property name="build" value="build"/>
  <property name="dist"  value="dist"/>

  <target name="init">
    <!-- Create the time stamp -->
    <tstamp/>
    <!-- Create the build directory structure used by compile -->
    <mkdir dir="${build}"/>
  </target>

  <target name="compile" depends="init">
    <!-- Compile the java code from ${src} into ${build} -->
    <javac srcdir="${src}" destdir="${build}"/>
  </target>

  <target name="dist" depends="compile">
    <!-- Create the distribution directory -->
    <mkdir dir="${dist}/lib"/>

    <!-- Put everything in ${build} into the MyProject-${DSTAMP}.jar file -->
    <jar jarfile="${dist}/lib/MyProject-${DSTAMP}.jar" basedir="${build}"/>
  </target>

  <target name="clean">
    <!-- Delete the ${build} and ${dist} directory trees -->
    <delete dir="${build}"/>
    <delete dir="${dist}"/>
  </target>
</project>
```

3.1.7 Token Filters

A project can have a set of tokens that might be automatically expanded if found when a file is copied, when the filtering-copy behavior is selected in the tasks that support this. These might be set in the build-file by the filter task [p 50] .

Since this can potentially be a very harmful behavior, the tokens in the files **must** be of the form @*token*@, where *token* is the token name that is set in the filter task. This token syntax matches the syntax of other build systems that perform such filtering and remains sufficiently orthogonal to most programming and scripting languages, as well as with documentation systems.

Note: If a token with the format @*token*@ is found in a file, but no filter is associated with that token, no changes take place; therefore, no escaping method is available – but as long as you choose appropriate names for your tokens, this should not cause problems.

Warning: If you copy binary files with filtering turned on, you can corrupt the files. This feature should be used with text files *only*.

3.1.8 Path-like Structures

You can specify PATH- and CLASSPATH-type references using both ":" and ";" as separator characters. Ant will convert the separator to the correct character of the current operating system.

Wherever path-like values need to be specified, a nested element can be used. This takes the general form of:

```
<classpath>
  <pathelement path="${classpath}"/>
  <pathelement location="lib/helper.jar"/>
</classpath>
```

The `location` attribute specifies a single file or directory relative to the project's base directory (or an absolute filename), while the `path` attribute accepts colon- or semicolon-separated lists of locations. The `path` attribute is intended to be used with predefined paths – in any other case, multiple elements with `location` attributes should be preferred.

As a shortcut, the `<classpath>` tag supports `path` and `location` attributes of its own, so:

```
<classpath>
  <pathelement path="${classpath}"/>
</classpath>
```

can be abreviated to:

```
<classpath path="${classpath}"/>
```

In addition, FileSet [p 20] s can be specified via nested `<fileset>` elements. The order in which the files building up a fileset are added to the path-like structure is not defined.

```
<classpath>
  <pathelement path="${classpath}"/>
  <fileset dir="lib">
    <include name="**/*.jar"/>
  </fileset>
  <pathelement location="classes"/>
</classpath>
```

Builds a path that holds the value of `${classpath}`, followed by all jar files in the `lib` directory, followed by the `classes` directory.

If you want to use the same path-like structure for several tasks, you can define them with a <path> element at the same level as *target*s, and reference them via their *id* attribute – see References [p 14] for an example.

A path-like structure can include a reference to another path-like structure via nested <path> elements:

```
<path id="base.path">
  <pathelement path="${classpath}"/>
  <fileset dir="lib">
    <include name="**/*.jar"/>
  </fileset>
  <pathelement location="classes"/>
</path>

<path id="tests.path">
  <path refid="base.path"/>
  <pathelement location="testclasses"/>
</path>
```

The shortcuts previously mentioned for <classpath> are also valid for <path>.For example:

```
<path id="base.path">
  <pathelement path="${classpath}"/>
</path>
```

can be written as:

```
<path id="base.path" path="${classpath}"/>
```

3.1.9 Command-line Arguments

Several tasks take arguments that will be passed to another process on the command line. To make it easier to specify arguments that contain space characters, nested arg elements can be used.

Attribute	Description	Required
value	a single command-line argument; can contain space characters.	Exactly one of these.
line	a space-delimited list of command-line arguments.	
file	The name of a file as a single command-line argument; will be replaced with the absolute filename of the file.	
path	A string that will be treated as a path-like string as a single command-line argument; you can use ; or : as path separators and Ant will convert it to the platform's local conventions.	

3.1.9.1 Examples

```
<arg value="-l -a"/>
```

is a single command-line argument containing a space character.

```
<arg line="-l -a"/>
```

represents two separate command-line arguments.

```
<arg path="/dir;/dir2:\dir3"/>
```

is a single command-line argument with the value `\dir;\dir2;\dir3` on DOS-based systems and `/dir:/dir2:/dir3` on Unix-like systems.

3.1.10 References

The `id` attribute of the buildfile's elements can be used to refer to them. This can useful if you are going to replicate the same snippet of XML over and over again – using a `<classpath>` structure more than once for example.

The following example:

```
<project ... >
  <target ... >
    <rmic ...>
      <classpath>
        <pathelement location="lib/"/>
        <pathelement path="${java.class.path}/"/>
        <pathelement path="${additional.path}"/>
      </classpath>
    </rmic>
  </target>

  <target ... >
    <javac ...>
      <classpath>
        <pathelement location="lib/"/>
        <pathelement path="${java.class.path}/"/>
        <pathelement path="${additional.path}"/>
      </classpath>
    </javac>
  </target>
</project>
```

could be rewritten as:

```
<project ... >
  <path id="project.class.path">
    <pathelement location="lib/"/>
    <pathelement path="${java.class.path}/"/>
    <pathelement path="${additional.path}"/>
  </path>

  <target ... >
    <rmic ...>
      <classpath refid="project.class.path"/>
    </rmic>
```

```
  </target>

  <target ... >
    <javac ...>
      <classpath refid="project.class.path"/>
    </javac>
  </target>
</project>
```

All tasks that use nested elements for PatternSet [p 19] s, FileSet [p 20] s or path-like structures [p 12] accept references to these structures as well.

4 Concepts and Types

4.1 Directory-based Tasks

Some tasks use directory trees for the task they perform. For example, the javac [p 63] task, which works upon a directory tree with `.java` files. Sometimes it can be very useful to work on a subset of that directory tree. This section describes how you can select a subset of such a directory tree.

Ant gives you two ways to create a subset, both of which can be used at the same time:

- Only include files/directories that match at least one pattern of a set of patterns.
- Exclude files/directories that match at least one pattern a set of patterns.

When both inclusion and exclusion are used, only files/directories that match the include patterns, and don't match the exclude patterns, are used.

Patterns can be specified inside the buildfile via task attributes or nested elements and via external files. Each line of the external file is taken as a pattern that is added to the list of include or exclude patterns.

4.1.1 Patterns

As described earlier, patterns are used for the inclusion and exclusion. These patterns look very much like the patterns used in DOS and UNIX:

'*' matches zero or more characters, '?' matches one character.

Examples:

`*.java` matches `.java`, `x.java` and `FooBar.java`, but not `FooBar.xml` (does not end with `.java`).

`?.java` matches `x.java`, `A.java`, but not `.java` or `xyz.java` (both don't have one character before `.java`).

Combinations of `*`'s and `?`'s are allowed.

Matching is done per-directory. This means that first the first directory in the pattern is matched against the first directory in the path to match. Then the second directory is matched, and so on. For example, when we have the pattern `/?abc/*/*.java` and the path `/xabc/foobar/test.java`, the first `?abc` is matched with `xabc`, then `*` is matched with `foobar`, and finally `*.java` is matched with `test.java`. They all match, so the path matches the pattern.

To make things a bit more flexible, we add one extra feature, which makes it possible to match multiple directory levels. This can be used to match a complete directory tree, or a file anywhere in the directory tree. To do this, `**` must be used as the name of a directory. When `**` is used as the name of a directory in the pattern, it matches zero or more directories. For example: `/test/**` matches all files/directories under `/test/`, such as `/test/x.java`, or `/test/foo/bar/xyz.html`, but not `/xyz.xml`.

There is one "shorthand" – if a pattern ends with / or \, then ** is appended. For example, mypackage/test/ is interpreted as if it were mypackage/test/**.

Example patterns:

**/CVS/*	Matches all files in CVS directories that can be located anywhere in the directory tree.
	Matches:
	`CVS/Repository`
	`org/apache/CVS/Entries`
	`org/apache/jakarta/tools/ant/CVS/Entries`
	But not:
	~~`org/apache/CVS/foo/bar/Entries (foo/bar/ part does not match)`~~

org/apache/jakarta/**	Matches all files in the org/apache/jakarta directory tree.
	Matches:
	`org/apache/jakarta/tools/ant/docs/index.html`
	`org/apache/jakarta/test.xml`
	But not:
	`org/apache/xyz.java`
	(jakarta/ part is missing).

org/apache/**/CVS/*	Matches all files in CVS directories that are located anywhere in the directory tree under org/apache.
	Matches:
	`org/apache/CVS/Entries`
	`org/apache/jakarta/tools/ant/CVS/Entries`
	But not:
	`org/apache/CVS/foo/bar/Entries`
	(foo/bar/ part does not match)

/test/	Matches all files that have a test element in their path, including test as a filename.

When these patterns are used in inclusion and exclusion, you have a powerful way to select just the files you want.

4.1.2 Examples

```
<copy todir="${dist}">
  <fileset dir="${src}"
           includes="**/images/*"
           excludes="**/*.gif"
  />
</copy>
```

This copies all files in directories called `images` that are located in the directory tree defined by `${src}` to the destination directory defined by `${dist}`, but excludes all `*.gif` files from the copy.

This example can also be expressed using nested elements:

```
<copy todir="${dist}">
  <fileset dir="${src}"/>
    <include name="**/images/*"/>
    <exclude name="**/*.gif"/>
  </fileset>
</copy>
```

4.1.3 Default Excludes

There are a set of definitions that are excluded by default from all directory-based tasks. They are:

```
**/*~
**/#*#
**/%*%
**/CVS
**/CVS/**
**/.cvsignore
```

If you do not want these default excludes applied, you may disable them with the `defaultexcludes="no"` attribute.

4.2 PatternSet

Patterns can be grouped to sets and later be referenced by their `id` attribute. They are defined via a `patternset` element, which can appear nested into a FileSet [p 20] or a directory-based task that constitutes an implicit FileSet. In addition, `patternset`s can be defined at the same level as `target` — i.e., as children of `project`.

Patterns can be specified by nested `<include>` or `<exclude>` elements or the following attributes.

Attribute	Description
includes	comma-separated list of patterns of files that must be included. All files are included when omitted.
includesfile	the name of a file; each line of this file is taken to be an include pattern.
excludes	comma-separated list of patterns of files that must be excluded; no files (except default excludes) are excluded when omitted.
excludesfile	the name of a file; each line of this file is taken to be an exclude pattern.

4.2.0.1 Examples

```
<patternset id="non.test.sources" >
  <include name="**/*.java"/>
  <exclude name="**/*Test*"/>
</patternset>
```

Builds a set of patterns that matches all `.java` files that do not contain the text `Test` in their name. This set can be referred [p 14] to via `<patternset refid="non.test.sources"/>`, by tasks that support this feature, or by FileSets.

Note that while the `includes` and `excludes` attributes accept multiple elements separated by commas or spaces, the nested <include> and <exclude> elements expect their name attribute to hold a single pattern.

The nested <include> and <exclude> elements allow you to use if and unless arguments to specify that the element should only be used if a property is set, or that it should be used only if a property is not set.

For example

```
<patternset id="sources" >
  <include name="std/**/*.java"/>
  <include name="prof/**/*.java" if="professional"/>
  <exclude name="**/*Test*"/>
</patternset>
```

will only include the files in the sub-directory *prof* if the property *professional* is set to some value.

4.3 FileSet

FileSets are groups of files. These files can be found in a directory tree starting in a base directory and are matched by patterns taken from a number of PatternSets [p 19] . FileSets can appear inside tasks that support this feature or at the same level as `target` – i.e., as children of `project`.

PatternSets can be specified as nested `<patternset>` elements. In addition, FileSet holds an implicit PatternSet and supports the nested `<include>` and `<exclude>` elements of PatternSet directly, as well as PatternSet's attributes.

Attribute	Description	Required
dir	the root of the directory tree of this FileSet.	Yes
defaultexcludes	indicates whether default excludes should be used or not (`yes | no`); default excludes are used when omitted.	No
includes	comma-separated list of patterns of files that must be included; all files are included when omitted.	No
includesfile	the name of a file; each line of this file is taken to be an include pattern.	No
excludes	comma-separated list of patterns of files that must be excluded; no files (except default excludes) are excluded when omitted.	No
excludesfile	the name of a file; each line of this file is taken to be an exclude pattern.	No

4.3.0.1 Examples

```
<fileset dir="${server.src}" >
  <patternset id="non.test.sources" >
    <include name="**/*.java"/>
    <exclude name="**/*Test*"/>
  </patternset>
</fileset>
```

Groups all files in directory `${server.src}` that are Java source files and don't have the text `Test` in their name.

```
<fileset dir="${client.src}" >
  <patternset refid="non.test.sources"/>
</fileset>
```

Groups all files in directory `${client.src}`, using the same patterns as the example above.

4.4 Mapping File Names

Some tasks take source files and create target files. Depending on the task, it may be quite obvious which name a target file will have (using javac [p 63] , you know there will be `.class` files for your `.java` files) – in other cases you may want to specify the target files, either to help Ant or to get an extra bit of functionality.

While source files are usually specified as fileset [p 20] s, you don't specify target files directly – instead, you tell Ant how to find the target file(s) for one source file. An instance of `org.apache.tools.ant.util.FileNameMapper` is responsible for this. It constructs target file names based on rules that can be parameterized with `from` and `to` attributes – the exact meaning of which is implementation-dependent.

These instances are defined in `<mapper>` elements with the following attributes:

Attribute	Description	Required
type	specifies one of the built-in implementations.	Exactly one of both
classname	specifies the implementation by class name.	
classpath	the classpath to use when looking up `classname`.	No
classpathref	the classpath to use, given as reference [p 14] to a path defined elsewhere.	No
from	the `from` attribute for the given implementation.	Depends on implementation.
to	the `to` attribute for the given implementation.	Depends on implementation.

The classpath can be specified via a nested `<classpath>`, as well – that is, a path [p 12] -like structure.

The built-in mapper types are:

4.4.0.1 identity

The target file name is identical to the source file name. Both `to` and `from` will be ignored.

Examples:

```
<mapper type="identity"/>
```

Source file name	Target file name
`A.java`	`A.java`
`foo/bar/B.java`	`foo/bar/B.java`
`C.properties`	`C.properties`
`Classes/dir/dir2/A.properties`	`Classes/dir/dir2/A.properties`

4.4.0.2 flatten

The target file name is identical to the source file name, with all leading directory information stripped off. Both `to` and `from` will be ignored.

Examples:

```
<mapper type="flatten"/>
```

Source file name	Target file name
A.java	A.java
foo/bar/B.java	B.java
C.properties	C.properties
Classes/dir/dir2/A.properties	A.properties

4.4.0.3 merge

The target file name will always be the same, as defined by `to` – `from` will be ignored.

4.4.0.3.1 Examples:

```
<mapper type="merge" to="archive.tar"/>
```

Source file name	Target file name
A.java	archive.tar
foo/bar/B.java	archive.tar
C.properties	archive.tar
Classes/dir/dir2/A.properties	archive.tar

4.4.0.4 glob

Both `to` and `from` define patterns that may contain at most one `*`. For each source file that matches the `from` pattern, a target file name will be constructed from the `to` pattern by substituting the `*` in the `to` pattern with the text that matches the `*` in the `from` pattern. Source file names that don't match the `from` pattern will be ignored.

Examples:

```
<mapper type="glob" from="*.java" to="*.java.bak"/>
```

Source file name	Target file name
A.java	A.java.bak
foo/bar/B.java	foo/bar/B.java.bak
C.properties	ignored
Classes/dir/dir2/A.properties	ignored

```
<mapper type="glob" from="C*ies" to="Q*y"/>
```

Source file name	Target file name
A.java	ignored
foo/bar/B.java	ignored
C.properties	Q.property
Classes/dir/dir2/A.properties	Qlasses/dir/dir2/A.property

4.4.0.5 regexp

Both `to` and `from` define regular expressions. If the source file name matches the `from` pattern, the target file name will be constructed from the `to` pattern, using `\0` to `\9` as back-references for the full match (`\0`) or the matches of the subexpressions in parentheses. Source files not matching the `from` pattern will be ignored.

Note that you need to escape a dollar-sign (`$`) with another dollar-sign in Ant.

The regexp mapper needs a supporting library and an implementation of `org.apache.tools.ant.util.regexp.RegexpMatcher` that hides the specifics of the library. Ant comes with implementations for jakarta-regexp (http://jakarta.apache.org/regexp/) and jakarta-ORO (http://jakarta.apache.org/oro/). If you compile from sources and plan to use one of them, make sure the libraries are in your `CLASSPATH`. For information about using gnu.regexp (http://www.cacas.org/~wes/java/) or gnu.rex (http://www.crocodile.org/~sts/Rex/) with Ant, see this (http://marc.theaimsgroup.com/?l=ant-devm;=97550753813481w;=2) article.

Ant will choose the regular-expression library based on the following algorithm:

- If the system property `ant.regexp.matcherimpl` has been set, it is taken as the name of the class implementing `org.apache.tools.ant.util.regexp.RegexpMatcher` that should be used.
- If it has not been set, first try jakarta-ORO; if that cannot be found, try jakarta-regexp.

Examples:

```
<mapper type="regexp" from="^(.*)\.java$$" to="\1.java.bak"/>
```

Source file name	Target file name
A.java	A.java.bak
foo/bar/B.java	foo/bar/B.java.bak
C.properties	ignored
Classes/dir/dir2/A.properties	ignored

```
<mapper type="regexp" from="^(.*)/([^/]+)/([^/]*)$$" to="\1/\2/\2-\3"/>
```

Source file name	Target file name
A.java	ignored
foo/bar/B.java	foo/bar/bar-B.java
C.properties	ignored
Classes/dir/dir2/A.properties	Classes/dir/dir2/dir2-A.properties

```
<mapper type="regexp" from="^(.*)\.(.*)$$" to="\2.\1"/>
```

Source file name	Target file name
A.java	java.A
foo/bar/B.java	java.foo/bar/B
C.properties	properties.C
Classes/dir/dir2/A.properties	properties.Classes/dir/dir2/A

5 Core Tasks

5.1 Ant

5.1.1 Description

Runs Ant on a supplied buildfile. This can be used to build subprojects.

When the *antfile* attribute is omitted, the file "build.xml" in the supplied directory (*dir* attribute) is used.

If no target attribute is supplied, the default target of the new project is used.

The properties of the current project will be available in the new project. These properties will override the properties that are set in the new project. (See also the property task [p 77]). You can set properties in the new project from the old project by using nested property tags. This allows you to parameterize your subprojects.

5.1.2 Parameters

Attribute	Description	Required
antfile	the buildfile to use. Defaults to "build.xml".	No
dir	the directory to use as a basedir for the new Ant project. Defaults to the current directory.	No
target	the target of the new Ant project that should be executed.	No
output	Filename to write the ant output to.	No

5.1.3 Examples

```
<ant antfile="subproject/subbuild.xml" dir="subproject" target="compile"/>

<ant dir="subproject"/>

<ant antfile="subproject/property_based_subbuild.xml">
  <property name="param1" value="version 1.x"/>
  <property file="config/subproject/default.properties"/>
</ant>
```

5.2 AntCall

5.2.1 Description

Call another target within the same build-file optionally specifying some properties (param's in this context)

5.2.2 Parameters

Attribute	Description	Required
target	The target to execute.	Yes

5.2.3 Parameters specified as nested elements

5.2.3.1 param

Specifies the properties to set before running the specified target. See property [p 77] for usage guidelines.

5.2.4 Examples

```
<target name="default">
  <antcall target="doSomethingElse">
    <param name="param1" value="value"/>
  </antcall>
</target>

<target name="doSomethingElse">
  <echo message="param1=${param1}"/>
</target>
```

Will run the target 'doSomethingElse' and echo 'param1=value'.

5.3 AntStructure

5.3.1 Description

Generates a DTD for Ant buildfiles which contains information about all tasks currently known to Ant.

Note that the DTD generated by this task is incomplete, you can always add XML entities using `<taskdef>` [p 98] . See here (http://www.sdv.fr/pages/casa/html/ant-dtd.en.html) for a way to get around this problem.

This task doesn't know about required attributes, all will be listed as `#IMPLIED`.

5.3.2 Parameters

Attribute	Description	Required
output	file to write the DTD to.	Yes

5.3.3 Examples

```
<antstructure output="project.dtd"/>
```

5.4 Apply

5.4.1 Description

Executes a system command. When the *os* attribute is specified, then the command is only executed when Ant is run on one of the specified operating systems.

The files and/or directories of a number of FileSet [p 20] s are passed as arguments to the system command. The timestamp of each source file is compared to the timestamp of a target file which is defined by a nested mapper [p 21] element. At least one fileset and exactly one mapper element are required.

5.4.2 Parameters

Attribute	Description	Required
executable	the command to execute without any command line arguments.	Yes
dest	the directory where the target files will be placed.	Yes
dir	the directory in which the command should be executed.	No
os	list of Operating Systems on which the command may be executed.	No
output	the file to which the output of the command should be redirected.	No
timeout	Stop the command if it doesn't finish within the specified time (given in milliseconds).	No
failonerror	Stop the buildprocess if the command exits with a returncode other than 0.	No
parallel	Run the command only once, appending all files as arguments. If false, command will be executed once for every file. Defaults to false.	No
type	One of *file*, *dir* or *both*. If set to *file*, only the names of plain files will be sent to the command. If set to *dir*, only the names of directories are considered.	No, default is *file*

5.4.3 Parameters specified as nested elements

5.4.3.1 fileset

You can use any number of nested `<fileset>` elements to define the files for this task and refer to `<fileset>`s defined elsewhere.

5.4.3.2 arg

Command line arguments should be specified as nested `<arg>` elements. See Command line arguments [p 13] .

5.4.3.3 srcfile

By default the file names of the source files will be added to the end of the command line. If you need to place it somewhere different, use a nested `<srcfile>` element between your `<arg>` elements to mark the insertion point.

5.4.3.4 targetfile

`<targetfile>` is similar to `<srcfile>` and marks the position of the target filename on the command line. If omitted, the target filenames will not be added to the command line at all.

5.4.3.5 env

It is possible to specify environment variables to pass to the system command via nested `<env>` elements. See the description in the section about exec [p 45]

Please note that the environment of the current Ant process is **not** passed to the system command if you specify variables using `<env>`.

5.4.4 Examples

```
<apply executable="cc" dest="src/C" parallel="false">
  <arg value="-c"/>
  <arg value="-o"/>
  <targetfile/>
  <srcfile/>
  <fileset dir="src/C" includes="*.c"/>
  <mapper type="glob" from="*.c" to="*.o"/>
</apply>
```

invokes `cc -c -o TARGETFILE SOURCEFILE` for each `.c` file that is newer than the corresponding `.o`, replacing TARGETFILE with the absolute filename of the `.o` and SOURCEFILE with the absolute name of the `.c` file.

5.5 Available

5.5.1 Description

Sets a property if a resource is available at runtime. This resource can be a file resource, a class in classpath or a JVM system resource.

If the resource is present, the property value is set to true by default, otherwise the property is not set. You can set the value to something specific by using the value attribute.

Normally, this task is used to set properties that are useful to avoid target execution depending on system parameters.

5.5.2 Parameters

Attribute	Description	Required
property	the name of the property to set.	Yes
value	the value to set the property to. Defaults to "true".	No
classname	the class to look for in classpath.	Yes
resource	the resource to look for in the JVM	
file	the file to look for.	
classpath	the classpath to use when looking up `classname`.	No
classpathref	the classpath to use, given as reference [p 14] to a path defined elsewhere.	No

5.5.3 Parameters specified as nested elements

5.5.3.1 classpath

`Available`'s *classpath* attribute is a path-like structure [p 12] and can also be set via a nested *classpath* element.

5.5.4 Examples

```
<available classname="org.whatever.Myclass" property="Myclass.present"/>
```

sets the property *`Myclass.present`* to the value "true" if the class *org.whatever.Myclass* is found in Ant's classpath.

```
<property name="jaxp.jar" value="./lib/jaxp11/jaxp.jar"/>
<available file="${jaxp.jar}" property="jaxp.jar.present"/>
```

sets the property *jaxp.jar.present* to the value "true" if the file *./lib/jaxp11/jaxp.jar* is found.

```
...in project ...
<property name="jaxp.jar" value="./lib/jaxp11/jaxp.jar"/>
<path id="jaxp" location="${jaxp.jar}"/>
...in target ...
<available classname="javax.xml.transform.Transformer" classpathref="jaxp" property="jaxp11.present"/>
```

sets the property *jaxp11.present* to the value "true" if the class *javax.xml.transform.Transformer* is found in the classpath referenced by `jaxp` (in this case, it is `./lib/jaxp11/jaxp.jar`).

5.6 Chmod

5.6.1 Description

Changes the permissions of a file or all files inside specified directories. Right now it has effect only under Unix. The permissions are also UNIX style, like the argument for the chmod command.

See the section on directory based tasks [p 16] , on how the inclusion/exclusion of files works, and how to write patterns.

This task holds an implicit FileSet [p 20] and supports all of FileSet's attributes and nested elements directly. More FileSets can be specified using nested `<fileset>` elements.

5.6.2 Parameters

Attribute	Description	Required
file	the file or single directory of which the permissions must be changed.	exactly one of the two or nested `<fileset>` elements.
dir	the directory which holds the files whose permissions must be changed.	
perm	the new permissions.	Yes
includes	comma separated list of patterns of files that must be included. All files are included when omitted.	No
includesfile	the name of a file. Each line of this file is taken to be an include pattern	No
excludes	comma separated list of patterns of files that must be excluded. No files (except default excludes) are excluded when omitted.	No
excludesfile	the name of a file. Each line of this file is taken to be an exclude pattern	No
defaultexcludes	indicates whether default excludes should be used or not ("yes"/"no"). Default excludes are used when omitted.	No
parallel	process all specified files using a single `chmod` command. Defaults to true.	No
type	One of *file*, *dir* or *both*. If set to *file*, only the permissions of plain files are going to be changed. If set to *dir*, only the directories are considered.	No, default is *file*

5.6.3 Examples

```
<chmod file="${dist}/start.sh" perm="ugo+rx"/>
```

makes the "start.sh" file readable and executable for anyone on a UNIX system.

```
    <chmod dir="${dist}/bin" perm="ugo+rx" includes="**/*.sh"/>
```

makes all ".sh" files below `${dist}/bin` readable and executable for anyone on a UNIX system.

```
<chmod perm="g+w">
  <fileset dir="shared/sources1">
    <exclude name="**/trial/**"/>
  </fileset>
  <fileset refid="other.shared.sources"/>
</chmod>
```

makes all files below `shared/sources1` (except those below any directory named trial) writable for members of the same group on a UNIX system. In addition all files belonging to a FileSet with `id other.shared.sources` get the same permissions.

5.7 Copy

5.7.1 Description

Copies a file or Fileset to a new file or directory. Files are only copied if the source file is newer than the destination file, or when the destination file does not exist. However, you can explicitly overwrite files with the *overwrite* attribute.

FileSet [p 20] s are used to select files to copy. To use a fileset, the *todir* attribute must be set.

5.7.2 Parameters

Attribute	Description	Required
file	The file to copy.	One of either *file* or at least one nested fileset element.
preservelastmodified	Give the copied files the same last modified time as the original files. Defaults to "no".	No
tofile	The file to copy to.	With the *file* attribute, either *tofile* or *todir* can be used. With nested filesets, only *todir* is allowed.
todir	The directory to copy to.	
overwrite	Overwrite existing files even if the destination files are newer. Defaults to "no".	No
filtering	Indicates whether token filtering should take place during the copy. Defaults to "no".	No
flatten	Ignore directory structure of source directory, copy all files into a single directory, specified by the *todir* attribute. Defaults to "no". Note that you can achieve the same effect by using a flatten mapper [p 22]	No
includeEmptyDirs	Copy empty directories included with the nested FileSet(s). Defaults to "yes".	No

5.7.3 Parameters specified as nested elements

5.7.3.1 mapper

You can define file name transformations by using a nested mapper [p 21] element. The default mapper used by `<copy>` is the identity [p 21] .

5.7.4 Examples

Copy a single file

```
<copy file="myfile.txt" tofile="mycopy.txt"/>
```

Copy a file to a directory

```
<copy file="myfile.txt" todir="../some/dir/tree"/>
```

Copy a directory to another directory

```
<copy todir="../new/dir">
  <fileset dir="src_dir"/>
</copy>
```

Copy a set of files to a directory

```
<copy todir="../dest/dir" >
  <fileset dir="src_dirquot; >
    <exclude name="**/*.java"/>
  </fileset>
</copy>

<copy todir="../dest/dir" >
  <fileset dir="src_dirquot; excludes="**/*.java"/>
</copy>
```

Copy a set of files to a directory appending "`.bak`" to the file name on the fly

```
<copy todir="../backup/dir" >
  <fileset dir="src_dirquot; />
  <mapper type="glob" from="*" to="*.bak"/>
</copy>
```

5.8 Copydir

5.8.1 Deprecated

This task has been deprecated. Use the Copy task instead.

5.8.2 Description

Copies a directory tree from the source to the destination.

It is possible to refine the set of files that are being copied. This can be done with the *includes*, *includesfile*, *excludes*, *excludesfile* and *defaultexcludes* attributes. With the *includes* or *includesfile* attribute you specify the files you want to have included by using patterns. The *exclude* or *excludesfile* attribute is used to specify the files you want to have excluded. This is also done with patterns. And finally with the *defaultexcludes* attribute, you can specify whether you want to use default exclusions or not. See the section on directory based tasks [p 16] , on how the inclusion/exclusion of files works, and how to write patterns.

This task forms an implicit FileSet [p 20] and supports all attributes of `<fileset>` (`dir` becomes `src`) as well as the nested `<include>`, `<exclude>` and `<patternset>` elements.

5.8.3 Parameters

Attribute	Description	Required
src	the directory to copy.	Yes
dest	the directory to copy to.	Yes
includes	comma separated list of patterns of files that must be included. All files are included when omitted.	No
includesfile	the name of a file. Each line of this file is taken to be an include pattern	No
excludes	comma separated list of patterns of files that must be excluded. No files (except default excludes) are excluded when omitted.	No
excludesfile	the name of a file. Each line of this file is taken to be an exclude pattern	No
defaultexcludes	indicates whether default excludes should be used or not ("yes"/"no"). Default excludes are used when omitted.	No
filtering	indicates whether token filtering should take place during the copy	No
flatten	ignore directory structure of source directory, copy all files into a single directory, specified by the `dest` attribute (default is `false`).	No
forceoverwrite	overwrite existing files even if the destination files are newer (default is false).	No

5.8.4 Examples

```
<copydir src="${src}/resources"
         dest="${dist}"
/>
```

copies the directory `${src}/resources` to `${dist}`.

```
<copydir src="${src}/resources"
         dest="${dist}"
         includes="**/*.java"
         excludes="**/Test.java"
/>
```

copies the directory `${src}/resources` to `${dist}` recursively. All java files are copied, except for files with the name `Test.java`.

```
<copydir src="${src}/resources"
         dest="${dist}"
         includes="**/*.java"
         excludes="mypackage/test/**"/>
```

copies the directory `${src}/resources` to `${dist}` recursively. All java files are copied, except for the files under the `mypackage/test` directory.

5.9 Copyfile

5.9.1 Deprecated

This task has been deprecated. Use the Copy task instead.

5.9.2 Description

Copies a file from the source to the destination. The file is only copied if the source file is newer than the destination file, or when the destination file does not exist.

5.9.3 Parameters

Attribute	Description	Required
src	the filename of the file to copy.	Yes
dest	the filename of the file where to copy to.	Yes
filtering	indicates whether token filtering should take place during the copy	No
forceoverwrite	overwrite existing files even if the destination files are newer (default is false).	No

5.9.4 Examples

```
<copyfile src="test.java" dest="subdir/test.java" />

<copyfile src="${src}/index.html" dest="${dist}/help/index.html" />
```

5.10 Cvs

5.10.1 Description

Handles packages/modules retrieved from a CVS (http://www.cvshome.org/) repository.

When doing automated builds, the get task [p 55] should be preferred over the *checkout* command, because of speed.

5.10.2 Parameters

Attribute	Description	Required
command	the CVS command to execute.	No, default "checkout"
cvsRoot	the CVSROOT variable.	No
dest	the directory where the checked out files should be placed.	No, default is project's basedir.
package	the package/module to check out.	No
tag	the tag of the package/module to check out.	No
date	Use the most recent revision no later than the given date	No
quiet	suppress informational messages.	No, default "false"
noexec	report only, don't change any files.	No, default "false"
output	the file to direct standard output from the command.	No, default output to ANT Log as MSG_INFO.
error	the file to direct standard error from the command.	No, default error to ANT Log as MSG_WARN.

5.10.3 Examples

```
<cvs cvsRoot=":pserver:anoncvs@jakarta.apache.org:/home/cvspublic"
     package="jakarta-tools"
     dest="${ws.dir}"
/>
```

checks out the package/module "jakarta-tools" from the CVS repository pointed to by the cvsRoot attribute, and stores the files in "${ws.dir}".

```
<cvs dest="${ws.dir}" command="update"/>
```

updates the package/module that has previously been checked out into "${ws.dir}".

```
<cvs command="-q diff -u -N" output="patch.txt"/>
```

silently (-q) creates a file called patch.txt which contains a unified (-u) diff which includes new files added via "cvs add" (-N) and can be used as input to patch.

```
<cvs command="update -A -d"/>
```

Updates from the head of repository ignoring sticky bits (-A) and creating any new directories as necessary (-d).

Note: the text of the command is passed to cvs "as-is" so any cvs options should appear before the command, and any command options should appear after the command as in the diff example above. See the cvs manual (http://www.cvshome.org/docs/manual/index.html) for details, specifically the Guide to CVS commands (http://www.cvshome.org/docs/manual/cvs_16.html)

5.11 Delete

5.11.1 Description

Deletes either a single file, all files in a specified directory and its sub-directories, or a set of files specified by one or more FileSet [p 20] s. When specifying a set of files, empty directories are *not* removed by default. To remove empty directories, use the *includeEmptyDirs* atribute.

5.11.2 Parameters

Attribute	Description	Required
file	The file to delete.	at least one of the two
dir	The directory to delete files from.	
verbose	Show name of each deleted file ("true"/"false"). Default is "false" when omitted.	No
quiet	If the file does not exist, do not display a diagnostic message or modify the exit status to reflect an error. This means that if a file or directory cannot be deleted, then no error is reported. This setting emulates the -f option to the Unix "rm" command. ("true"/"false"). Default is "false" meaning things are "noisy".	No
includeEmptyDirs	Set to "true" to delete empty directories when using filesets. Default is "false".	No
includes	*Deprecated.* Comma separated list of patterns of files that must be deleted. All files are in the current directory and any sub-directories are deleted when omitted.	No
includesfile	*Deprecated.* The name of a file. Each line of this file is taken to be an include pattern	No
excludes	*Deprecated.* Comma separated list of patterns of files that must be excluded from the deletion list. No files (except default excludes) are excluded when omitted.	No
excludesfile	*Deprecated.* The name of a file. Each line of this file is taken to be an exclude pattern	No
defaultexcludes	*Deprecated.* Indicates whether default excludes should be used or not ("yes"/"no"). Default excludes are used when omitted.	No

5.11.3 Examples

```
<delete file="/lib/ant.jar"/>
```

deletes the file `/lib/ant.jar`.

```
<delete dir="lib"/>
```

deletes the `lib` directory, including all files and subdirectories of `lib`.

```
<delete>
  <fileset dir="." includes="**/*.bak"/>
</delete>
```

deletes all files with the extension "`.bak`quot; from the current directory and any sub-directories.

```
<delete includeEmptyDirs="true" >
  <fileset dir="build" />
</delete>
```

deletes all files and subdirectories of `build`, but not `build` itself.

5.12 Deltree

5.12.1 Deprecated

This task has been deprecated. Use the Delete task instead.

5.12.2 Description

Deletes a directory with all its files and subdirectories.

5.12.3 Parameters

Attribute	Description	Required
dir	the directory to delete.	Yes

5.12.4 Examples

```
<deltree dir="dist"/>
```

deletes the directory `dist`, including its files and subdirectories.

```
<deltree dir="${dist}"/>
```

deletes the directory `${dist}`, including its files and subdirectories.

5.13 Echo

5.13.1 Description

Echoes a message to System.out or a file.

5.13.2 Parameters

Attribute	Description	Required
message	the message to echo.	Yes, unless data is included in a character section within this element.
file	the file to write the message to.	No
append	Append to an existing file?	No – default is false.

5.13.3 Examples

```
  <echo message="Hello world"/>


<echo>
This is a longer message stretching over
two lines.
</echo>
```

5.14 Exec

5.14.1 Description

Executes a system command. When the *os* attribute is specified, then the command is only executed when Ant is run on one of the specified operating systems.

5.14.2 Parameters

Attribute	Description	Required
command	the command to execute with all command line arguments. **deprecated, use executable and nested `<arg>` elements instead**.	Exactly one of the two.
executable	the command to execute without any command line arguments.	
dir	the directory in which the command should be executed.	No
os	list of Operating Systems on which the command may be executed.	No
output	the file to which the output of the command should be redirected.	No
timeout	Stop the command if it doesn't finish within the specified time (given in milliseconds).	No
failonerror	Stop the buildprocess if the command exits with a returncode other than 0.	No
newenvironment	Do not propagate old environment when new environment variables are specified.	No, default is *false*

5.14.3 Examples

```
<exec dir="${src}" executable="cmd.exe" os="Windows 2000" output="dir.txt">
  <arg line="/c dir"/>
</exec>
```

5.14.4 Parameters specified as nested elements

5.14.4.1 arg

Command line arguments should be specified as nested `<arg>` elements. See Command line arguments [p 13] .

5.14.4.2 env

It is possible to specify environment variables to pass to the system command via nested `<env>` elements.

Please note that the environment of the current Ant process is **not** passed to the system command if you specify variables using `<env>`.

Attribute	Description	Required
key	The name of the environment variable.	Yes
value	The literal value for the environment variable.	Exactly one of these.
path	The value for a PATH like environment variable. You can use ; or : as path separators and Ant will convert it to the platform's local conventions.	
file	The value for the environment variable. Will be replaced by the absolute file-name of the file by Ant.	

5.14.4.2.1 Examples

```
<exec executable="emacs" >
  <env key="DISPLAY" value=":1.0"/>
</exec>
```

starts `emacs` on display 1 of the X Window System.

```
<exec ... >
  <env key="PATH" path="${java.library.path}:${basedir}/bin"/>
</exec>
```

adds `${basedir}/bin` to the `PATH` of the system command.

Note: Although it may work for you to specify arguments using a simple arg-element and separate them by spaces it may fail if you switch to a newer version of the JDK. JDK < 1.2 will pass these as separate arguments to the program you are calling, JDK >= 1.2 will pass them as a single argument and cause most calls to fail.

Note2: If you are using Ant on Windows and a new DOS-Window pops up for every command which is executed this may be a problem of the JDK you are using. This problem may occur with all JDK's < 1.2.

5.15 ExecOn

5.15.1 Description

Executes a system command. When the *os* attribute is specified, then the command is only executed when Ant is run on one of the specified operating systems.

The files and/or directories of a number of FileSet [p 20] s are passed as arguments to the system command. At least one nested `<fileset>` is required.

5.15.2 Parameters

Attribute	Description	Required
executable	the command to execute without any command line arguments.	Yes
dir	the directory in which the command should be executed.	No
os	list of Operating Systems on which the command may be executed.	No
output	the file to which the output of the command should be redirected.	No
timeout	Stop the command if it doesn't finish within the specified time (given in milliseconds).	No
failonerror	Stop the buildprocess if the command exits with a returncode other than 0.	No
newenvironment	Do not propagate old environment when new environment variables are specified.	No, default is *false*
parallel	Run the command only once, appending all files as arguments. If false, command will be executed once for every file. Defaults to false.	No
type	One of *file*, *dir* or *both*. If set to *file*, only the names of plain files will be sent to the command. If set to *dir*, only the names of directories are considered.	No, default is *file*

5.15.3 Parameters specified as nested elements

5.15.3.1 fileset

You can use any number of nested `<fileset>` elements to define the files for this task and refer to `<fileset>`s defined elsewhere.

5.15.3.2 arg

Command line arguments should be specified as nested `<arg>` elements. See Command line arguments [p 13] .

5.15.3.3 srcfile

By default the file names of the source files will be added to the end of the command line. If you need to place it somewhere different, use a nested `<srcfile>` element between your `<arg>` elements to mark the insertion point.

5.15.3.4 env

It is possible to specify environment variables to pass to the system command via nested `<env>` elements. See the description in the section about exec [p 45]

Please note that the environment of the current Ant process is **not** passed to the system command if you specify variables using `<env>`.

5.15.4 Examples

```
<execon executable="ls" >
  <arg value="-l"/>
  <fileset dir="/tmp">
    <patternset>
      <exclude name="**/*.txt"/>
    </patternset>
  </fileset>
  <fileset refid="other.files"/>
</execon>
```

invokes `ls -l`, adding the absolute filenames of all files below `/tmp` not ending in `.txt` and all files of the FileSet with `id other.files` to the command line.

```
<execon executable="somecommand" parallel="false" >
  <arg value="arg1"/>
  <srfile/>
  <arg value="arg2"/>
  <fileset dir="/tmp"/>
</execon>
```

invokes `somecommand arg1 SOURCEFILENAME arg2` for each file in `/tmp` replacing SOURCEFILENAME with the absolute filename of each file in turn. If `parallel` had been set to true, SOURCEFILENAME would be replaced with the absolute filenames of all files separated by spaces.

5.16 Fail

5.16.1 Description

Exits the current build (just throwing a BuildException), optionally printing additional information.

5.16.2 Parameters

Attribute	Description	Required
message	A message giving further information on why the build exited	No

5.16.3 Examples

```
  <fail/>
```

will exit the current build with no further information given.

```
BUILD FAILED

build.xml:4: No message
```

```
  <fail message="Something wrong here."/>
```

will exit the current build and print something like the following to wherever your output goes:

```
BUILD FAILED

build.xml:4: Something wrong here.
```

5.17 Filter

5.17.1 Description

Sets a token filter for this project or read multiple token filter from an input file and sets these as filters. Token filters are used by all tasks that perform file copying operations through the Project commodity methods.

Note 1: the token string must not contain the separators chars (@).
Note 2: Either token and value attributes must be provided, or only the filtersfile attribute.

5.17.2 Parameters

Attribute	Description	Required
token	the token string without @	Yes*
value	the string that should be put to replace the token when the file is copied	Yes*
filtersfile	The file from which the filters must be read. This file must be a formatted as a property file.	Yes*

* see notes 1 and 2 above parameters table.

5.17.3 Examples

```
<filter token="year" value="2000"/>
<copy todir="${dest.dir}" filtering="true">
  <fileset dir="${src.dir}"/>
</copy>
```

will copy recursively all the files from the *src.dir* directory into the *dest.dir* directory replacing all the occurences of the string *@year@* with *2000.*

```
<filter filtersfile="deploy_env.properties"/>
```

will read all property entries from the *deploy_env.properties* file and set these as filters.

5.18 FixCRLF

5.18.1 Description

Adjusts a text file to local.

It is possible to refine the set of files that are being adjusted. This can be done with the *includes*, *includesfile*, *excludes*, *excludesfile* and *defaultexcludes* attributes. With the *includes* or *includesfile* attribute you specify the files you want to have included by using patterns. The *exclude* or *excludesfile* attribute is used to specify the files you want to have excluded. This is also done with patterns. And finally with the *defaultexcludes* attribute, you can specify whether you want to use default exclusions or not. See the section on directory based tasks [p 16] , on how the inclusion/exclusion of files works, and how to write patterns.

This task forms an implicit FileSet [p 20] and supports all attributes of `<fileset>` (`dir` becomes `srcdir`) as well as the nested `<include>`, `<exclude>` and `<patternset>` elements.

5.18.2 Parameters

Attribute	Description	Required
srcDir	Where to find the files to be fixed up.	Yes
destDir	Where to place the corrected files. Defaults to srcDir (replacing the original file)	No
includes	comma separated list of patterns of files that must be included. All files are included when omitted.	No
includesfile	the name of a file. Each line of this file is taken to be an include pattern	No
excludes	comma separated list of patterns of files that must be excluded. No files (except default excludes) are excluded when omitted.	No
excludesfile	the name of a file. Each line of this file is taken to be an exclude pattern	No
defaultexcludes	indicates whether default excludes should be used or not ("yes"/"no"). Default excludes are used when omitted.	No
cr	Specifies how carriage return (CR) characters are to be handled. Valid values for this property are: • add: ensure that there is a CR before every LF • asis: leave CR characters alone • remove: remove all CR characters Default is based on the platform on which you are running this task. For Unix platforms, the default is remove. For DOS based systems (including Windows), the default is add. Note: Unless this property is specified as "asis", extra CR characters which do not precede a LF will be removed.	No

tab	Specifies how tab characters are to be handled. Valid values for this property are: • add: convert sequences of spaces which span a tab stop to tabs • asis: leave tab and space characters alone • remove: convert tabs to spaces Default for this parameter is "asis". Note: Unless this property is specified as "asis", extra spaces and tabs after the last non-whitespace character on the line will be removed.	No
tablength	The number of characters a TAB stop corresponds to. Must be a positive power of 2, default for this parameter is 8.	No
eof	Specifies how DOS end of file (control-Z) characters are to be handled. Valid values for this property are: • add: ensure that there is an EOF character at the end of the file • asis: leave EOF characters alone • remove: remove any EOF character found at the end Default is based on the platform on which you are running this task. For Unix platforms, the default is remove. For DOS based systems (including Windows), the default is asis.	No

5.18.3 Examples

```
<fixcrlf srcdir="${src}"
     cr="remove" eof="remove"
     includes="**/*.sh"
/>
```

Removes carriage return and eof characters from the shell scripts. Tabs and spaces are left as is.

```
<fixcrlf srcdir="${src}"
     cr="add"
     includes="**/*.bat"
/>
```

Ensures that there are carriage return characters prior to evey line feed. Tabs and spaces are left as is. EOF characters are left alone if run on DOS systems, and are removed if run on Unix systems.

```
<fixcrlf srcdir="${src}"
     tabs="add"
     includes="**/Makefile"
/>
```

Adds or removes CR characters to match local OS conventions, and converts spaces to tabs when appropriate. EOF characters are left alone if run on DOS systems, and are removed if run on Unix systems. Many versions of make require tabs prior to commands.

```
<fixcrlf srcdir="${src}"
     tabs="remove"
     includes="**/README*"
/>
```

Adds or removes CR characters to match local OS conventions, and converts all tabs to spaces. EOF characters are left alone if run on DOS systems, and are removed if run on Unix systems. You never know what editor a user will use to browse README's.

5.19 GenKey

5.19.1 Description

Generates a key in keystore.

5.19.2 Parameters

Attribute	Description	Required
alias	the alias to add under	Yes.
storepass	password for keystore integrity.	Yes.
keystore	keystore location	No
storetype	keystore type	No
keypass	password for private key (if different)	No
sigalg	the algorithm to use in signing	No
keyalg	the method to use when generating name-value pair	No
verbose	(true | false) verbose output when signing	No
dname	The distinguished name for entity	Yes if dname element unspecified
validity	(integer) indicates how many days certificate is valid	No
keysize	(integer) indicates the size of key generated	No

Alternatively you can specify the distinguished name by creating a sub-element named dname and populating it with param elements that have a name and a value. When using the subelement it is automatically encoded properly and , are replace

The following two examples are identical:

5.19.3 Examples

```
<genkey alias="apache-group" storepass="secret" dname="CN=Ant Group,
OU=Jakarta Division, O=Apache.org, C=US"/>

<genkey alias="apache-group" storepass="secret" >
  <dname>
    <param name="CN" value="Ant Group"/>
    <param name="OU" value="Jakarta Division"/>
    <param name="O"  value="Apache.Org"/>
    <param name="C"  value="US"/>
  </dname>
</genkey>
```

5.20 Get

5.20.1 Description

Gets a file from a URL. When the verbose option is "on", this task displays a '.' for every 100 Kb retrieved.

This task should be preferred above the CVS task [p 39] when doing automated builds. CVS is significantly slower than loading a compressed archive with http/ftp.

The *usetimestamp* option enables you to control downloads so that the remote file is only fetched if newer than the local copy. If there is no local copy, the download always takes place. When a file is downloaded, the timestamp of the downloaded file is set to the remote timestamp, if the JVM is Java1.2 or later. NB: This timestamp facility only works on downloads using the HTTP protocol.

5.20.2 Parameters

Attribute	Description	Required
src	the URL from which to retrieve a file.	Yes
dest	the file where to store the retrieved file.	Yes
verbose	show verbose progress information ("on"/"off").	No
ignoreerrors	Log errors but don't treat as fatal.	No
usetimestamp	conditionally download a file based on the timestamp of the local copy. HTTP only	No

5.20.3 Examples

```
<get src="http://jakarta.apache.org/" dest="help/index.html"/>
```

Gets the index page of http://jakarta.apache.org/, and stores it in the file `help/index.html`.

```
<get src="http://jakarta.apache.org/builds/tomcat/nightly/ant.zip"
      dest="optional.jar"
      verbose="true"
      usetimestamp="true"/>
```

Gets the nightly ant build from the tomcat distribution, if the local copy is missing or out of date. Uses the verbose option for progress information.

5.21 GUnzip

5.21.1 Description

Expands a GZip file.

If *dest* is a directory the name of the destination file is the same as *src* (with the ".gz" extension removed if present). If *dest* is omitted, the parent dir of *src* is taken. The file is only expanded if the source file is newer than the destination file, or when the destination file does not exist.

5.21.2 Parameters

Attribute	Description	Required
src	the file to expand.	Yes
dest	the destination file or directory.	No

5.21.3 Examples

```
<gunzip src="test.tar.gz"/>
```

expands *test.tar.gz* to *test.tar*

```
<gunzip src="test.tar.gz" dest="test2.tar"/>
```

expands *test.tar.gz* to *test2.tar*

```
<gunzip src="test.tar.gz" dest="subdir"/>
```

expands *test.tar.gz* to *subdir/test.tar* (assuming subdir is a directory).

5.22 GZip

5.22.1 Description

GZips a file.

5.22.2 Parameters

Attribute	Description	Required
src	the file to gzip.	Yes
zipfile	the destination file.	Yes

5.22.3 Examples

```
<gzip src="test.tar" zipfile="test.tar.gz" />
```

5.23 Jar

5.23.1 Description

Jars a set of files.

The *basedir* attribute is the reference directory from where to jar.

Note that file permissions will not be stored in the resulting jarfile.

It is possible to refine the set of files that are being jarred. This can be done with the *includes*, *includesfile*, *excludes*, *excludesfile* and *defaultexcludes* attributes. With the *includes* or *includesfile* attribute you specify the files you want to have included by using patterns. The *exclude* or *excludesfile* attribute is used to specify the files you want to have excluded. This is also done with patterns. And finally with the *default-excludes* attribute, you can specify whether you want to use default exclusions or not. See the section on directory based tasks [p 16] , on how the inclusion/exclusion of files works, and how to write patterns.

This task forms an implicit FileSet [p 20] and supports all attributes of `<fileset>` (`dir` becomes `basedir`) as well as the nested `<include>`, `<exclude>` and `<patternset>` elements.

You can also use nested file sets for more flexibility, and specify multiple ones to merge together different trees of files into one JAR. The extended fileset attributes from the zip task are also available in the jar task. See the Zip [p 108] task for more details and examples.

If the manifest is omitted, a simple one will be supplied by Ant. You should not include `META-INF/MANIFEST.MF` in your set of files.

The `whenempty` parameter controls what happens when no files match. If `create` (the default), the JAR is created anyway with only a manifest. If `skip`, the JAR is not created and a warning is issued. If `fail`, the JAR is not created and the build is halted with an error.

(The Jar task is a shortcut for specifying the manifest file of a JAR file. The same thing can be accomplished by using the *fullpath* attribute of a zipfileset in a Zip task. The one difference is that if the *manifest* attribute is not specified, the Jar task will include an empty one for you.)

5.23.2 Parameters

Attribute	Description	Required
jarfile	the jar-file to create.	Yes
basedir	the directory from which to jar the files.	No
compress	Not only store data but also compress them, defaults to true	No
includes	comma separated list of patterns of files that must be included. All files are included when omitted.	No
includesfile	the name of a file. Each line of this file is taken to be an include pattern	No
excludes	comma separated list of patterns of files that must be excluded. No files (except default excludes) are excluded when omitted.	No
excludesfile	the name of a file. Each line of this file is taken to be an exclude pattern	No
defaultexcludes	indicates whether default excludes should be used or not ("yes"/"no"). Default excludes are used when omitted.	No
manifest	the manifest file to use.	No
whenempty	Behavior to use if no files match.	No

5.23.3 Examples

```
<jar jarfile="${dist}/lib/app.jar" basedir="${build}/classes"/>
```

jars all files in the `${build}/classes` directory into a file called `app.jar` in the `${dist}/lib` directory.

```
<jar jarfile="${dist}/lib/app.jar"
     basedir="${build}/classes"
     excludes="**/Test.class"
/>
```

jars all files in the `${build}/classes` directory into a file called `app.jar` in the `${dist}/lib` directory. Files with the name `Test.class` are excluded.

```
<jar jarfile="${dist}/lib/app.jar"
     basedir="${build}/classes"
     includes="mypackage/test/**"
     excludes="**/Test.class"
/>
```

jars all files in the `${build}/classes` directory into a file called `app.jar` in the `${dist}/lib` directory. Only files under the directory `mypackage/test` are used, and files with the name `Test.class` are excluded.

```
<jar jarfile="${dist}/lib/app.jar">
  <fileset dir="${build}/classes"
           excludes="**/Test.class"
  />
  <fileset dir="${src}/resources"/>
</jar>
```

jars all files in the `${build}/classes` directory and also in the `${src}/resources` directory together into a file called `app.jar` in the `${dist}/lib` directory. Files with the name `Test.class` are excluded. If there are files such as `${build}/classes/mypackage/MyClass.class` and `${src}/resources/mypackage/image.gif`, they will appear in the same directory in the JAR (and thus be considered in the same package by Java).

5.24 Java

5.24.1 Description

Executes a Java class within the running (Ant) VM or forks another VM if specified.

Be careful that the executed class doesn't call System.exit(), because it will terminate the VM and thus Ant. In case this happens, it's highly suggested that you set the fork attribute so that System.exit() stops the other VM and not the one that is currently running Ant.

5.24.2 Parameters

Attribute	Description	Required
classname	the Java class to execute.	Yes
args	the arguments for the class that is executed. **deprecated, use nested `<arg>` elements instead.**	No
classpath	the classpath to use.	No
classpathref	the classpath to use, given as reference [p 14] to a PATH defined elsewhere.	No
fork	if enabled triggers the class execution in another VM (disabled by default)	No
jvm	the command used to invoke the Java Virtual Machine, default is 'java'. The command is resolved by java.lang.Runtime.exec(). Ignored if fork is disabled.	No
jvmargs	the arguments to pass to the forked VM (ignored if fork is disabled). **deprecated, use nested `<jvmarg>` elements instead.**	No
maxmemory	Max amount of memory to allocate to the forked VM (ignored if fork is disabled)	No
failonerror	Stop the buildprocess if the command exits with a returncode other than 0. Only available if fork is true.	No
dir	The directory to invoke the VM in. (ignored if fork is disabled)	No
output	Name of a file to write the output to.	No

5.24.3 Parameters specified as nested elements

5.24.3.1 arg and jvmarg

Use nested `<arg>` and `<jvmarg>` elements to specify arguments for the or the forked VM. See Command line arguments [p 13] .

5.24.3.2 sysproperty

Use nested `<sysproperty>` elements to specify system properties required by the class. These properties will be made available to the VM during the execution of the class (either ANT's VM or the forked VM). The attributes for this element are the same as for environment variables [p 45] .

5.24.3.3 classpath

`Java`'s *classpath* attribute is a PATH like structure [p 12] and can also be set via a nested *classpath* element.

5.24.3.3.1 Example

```
<java classname="test.Main" >
  <arg value="-h"/>
  <classpath>
    <pathelement location="\test.jar"/>
    <pathelement path="${java.class.path}"/>
  </classpath>
</java>
```

5.24.4 Examples

```
<java classname="test.Main"/>

<java classname="test.Main"
      fork="yes" >
  <sysproperty key="DEBUG" value="true"/>
  <arg value="-h"/>
  <jvmarg value="-Xrunhprof:cpu=samples,file=log.txt,depth=3"/>
</java>
```

5.25 Javac

5.25.1 Description

Compiles a source tree within the running (Ant) VM.

The source and destination directory will be recursively scanned for Java source files to compile. Only Java files that have no corresponding class file or where the class file is older than the java file will be compiled.

The directory structure of the source tree should follow the package hierarchy.

It is possible to refine the set of files that are being compiled/copied. This can be done with the *includes*, *includesfile*, *excludes*, *excludesfile* and *defaultexcludes* attributes. With the *includes* or *includesfile* attribute you specify the files you want to have included by using patterns. The *exclude* or *excludesfile* attribute is used to specify the files you want to have excluded. This is also done with patterns. And finally with the *defaultexcludes* attribute, you can specify whether you want to use default exclusions or not. See the section on directory based tasks [p 16] , on how the inclusion/exclusion of files works, and how to write patterns.

It is possible to use different compilers. This can be selected with the "build.compiler" property. There are four choices:

- classic (the standard compiler of JDK 1.1/1.2)
- modern (the new compiler of JDK 1.3)
- jikes (the Jikes (http://oss.software.ibm.com/developerworks/opensource/jikes/project) compiler)
- jvc (the Command-Line Compiler from Microsoft's SDK for Java / Visual J++)

For JDK 1.1/1.2, classic is the default. For JDK 1.3, modern is the default. If you wish to use a different compiler interface than one of the four supplied, write a class that implements the CompilerAdapter interface (package org.apache.tools.ant.taskdefs.compilers). Supply the full classname in the "build.compiler" property.

5.25.2 Parameters

Attribute	Description	Required
srcdir	location of the java files. (See Notes at the end)	Yes, unless nested `<src>` elements are present.
destdir	location to store the class files.	No
includes	comma-separated list of patterns of files that must be included; all files are included when omitted.	No
includesfile	the name of a file that contains include patterns.	No

excludes	comma-separated list of patterns of files that must be excluded; no files (except default excludes) are excluded when omitted.	No
excludesfile	the name of a file that contains exclude patterns.	No
defaultexcludes	indicates whether default excludes should be used (`yes` | `no`); default excludes are used when omitted.	No
classpath	the classpath to use.	No
bootclasspath	location of bootstrap class files.	No
classpathref	the classpath to use, given as a reference [p 14] to a PATH defined elsewhere.	No
bootclasspathref	location of bootstrap class files, given as a reference [p 14] to a PATH defined elsewhere.	No
extdirs	location of installed extensions.	No
encoding	encoding of source files.	No
debug	indicates whether source should be compiled with debug information; defaults to `off`.	No
optimize	indicates whether source should be compiled with optimization; defaults to `off`.	No
deprecation	indicates whether source should be compiled with deprecation information; defaults to `off`.	No
target	generate class files for specific VM version (e.g., `1.1` or `1.2`).	No
verbose	asks the compiler for verbose output.	No
depend	enables dependency-tracking for compilers that support this (jikes and classic)	No
includeAntRuntime	whether to include the Ant run-time libraries; defaults to `yes`.	No
includeJavaRuntime	whether to include the default run-time libraries from the executing VM; defaults to `no`.	No
failonerror	indicates whether the build will continue even if there are compilation errors; defaults to `true`.	No

5.25.3 Parameters specified as nested elements

This task forms an implicit FileSet [p 20] and supports all attributes of `<fileset>` (`dir` becomes `srcdir`) as well as the nested `<include>`, `<exclude>` and `<patternset>` elements.

5.25.3.1 src, classpath, bootclasspath and extdirs

Javac's *srcdir*, *classpath*, *bootclasspath* and *extdirs* attributes are path-like structures [p 12] and can also be set via nested <srcgt;, <classpathgt;, <bootclasspathgt; and <extdirsgt; elements, respectively.

5.25.4 Examples

```
<javac srcdir="${src}"
       destdir="${build}"
       classpath="xyz.jar"
       debug="on"
/>
```

compiles all .java files under the ${src} directory, and stores the .class files in the ${build} directory. The classpath used contains xyz.jar, and debug information is on.

```
<javac srcdir="${src}"
       destdir="${build}"
       includes="mypackage/p1/**,mypackage/p2/**"
       excludes="mypackage/p1/testpackage/**"
       classpath="xyz.jar"
       debug="on"
/>
```

compiles .java files under the ${src} directory, and stores the .class files in the ${build} directory. The classpath used contains xyz.jar, and debug information is on. Only files under mypackage/p1 and mypackage/p2 are used. Files in the mypackage/p1/testpackage directory are excluded from compilation.

```
<javac srcdir="${src}:${src2}"
       destdir="${build}"
       includes="mypackage/p1/**,mypackage/p2/**"
       excludes="mypackage/p1/testpackage/**"
       classpath="xyz.jar"
       debug="on"
/>
```

is the same as the previous example, with the addition of a second source path, defined by the property src2. This can also be represented using nested <src> elements as follows:

```
<javac destdir="${build}"
       classpath="xyz.jar"
       debug="on">
  <src path="${src}"/>
  <src path="${src2}"/>
  <include name="mypackage/p1/**"/>
  <include name="mypackage/p2/**"/>
  <exclude name="mypackage/p1/testpackage/**"/>
</javac>
```

Note: If you are using Ant on Windows and a new DOS window pops up for every use of an external compiler, this may be a problem of the JDK you are using. This problem may occur with all JDKs < 1.2.

Note: If you wish to compile only source-files located in some packages below a common root you should not include these packages in the srcdir-attribute. Use include/exclude-attributes or elements to filter for these packages. If you include part of your package-structure inside the srcdir-attribute (or nested src-elements) Ant will start to recompile your sources everytime you call it.

5.26 Javadoc/Javadoc2

5.26.1 Description

Generates code documentation using the javadoc tool.

The source directory will be recursively scanned for Java source files to process but only those matching the inclusion rules, and not matching the exclusions rules will be passed to the javadoc tool. This allows wildcards to be used to choose between package names, reducing verbosity and management costs over time. This task, however, has no notion of "changed" files, unlike the javac [p 63] task. This means all packages will be processed each time this task is run. In general, however, this task is used much less frequently.

This task works seamlessly between different javadoc versions (1.1 and 1.2), with the obvious restriction that the 1.2 attributes will be ignored if run in a 1.1 VM.

NOTE: since javadoc calls System.exit(), javadoc cannot be run inside the same VM as ant without breaking functionality. For this reason, this task always forks the VM. This overhead is not significant since javadoc is normally a heavy application and will be called infrequently.

NOTE: the packagelist attribute allows you to specify the list of packages to document outside of the Ant file. It's a much better practice to include everything inside the build.xml file. This option was added in order to make it easier to migrate from regular makefiles, where you would use this option of javadoc. The packages listed in packagelist are not checked, so the task performs even if some packages are missing or broken. Use this option if you wish to convert from an existing makefile. Once things are running you should then switch to the regular notation.

DEPRECATION: the javadoc2 task simply points to the javadoc task and it's there for back compatibility reasons. Since this task will be removed in future versions, you are strongly encouraged to use javadoc [p 67] instead.

5.26.2 Parameters

Attribute	Description	Availability	Required
sourcepath	Specify where to find source files	all	At least one of the two or nested `<sourcepath>`
sourcepathref	Specify where to find source files by reference [p 14] to a PATH defined elsewhere.	all	
destdir	Destination directory for output files	all	Yes
maxmemory	Max amount of memory to allocate to the javadoc VM	all	No
sourcefiles	Comma separated list of source files	all	at least one of the two
packagenames	Comma separated list of package files (with terminating wildcard)	all	

packageList	The name of a file containing the packages to process	all	No
classpath	Specify where to find user class files	all	No
Bootclasspath	Override location of class files loaded by the bootstrap class loader	1.2	No
classpathref	Specify where to find user class files by reference [p 14] to a PATH defined elsewhere.	all	No
bootclasspathref	Override location of class files loaded by the bootstrap class loader by reference [p 14] to a PATH defined elsewhere.	1.2	No
Extdirs	Override location of installed extensions	1.2	No
Overview	Read overview documentation from HTML file	1.2	No
Public	Show only public classes and members	all	No
Protected	Show protected/public classes and members (default)	all	No
Package	Show package/protected/public classes and members	all	No
Private	Show all classes and members	all	No
Old	Generate output using JDK 1.1 emulating doclet	1.2	No
Verbose	Output messages about what Javadoc is doing	1.2	No
Locale	Locale to be used, e.g. en_US or en_US_WIN	1.2	No
Encoding	Source file encoding name	all	No
Version	Include @version paragraphs	all	No
Use	Create class and package usage pages	1.2	No
Author	Include @author paragraphs	all	No
Splitindex	Split index into one file per letter	1.2	No
Windowtitle	Browser window title for the documentation (text)	1.2	No
Doctitle	Include title for the package index(first) page (html-code)	1.2	No
Header	Include header text for each page (html-code)	1.2	No
Footer	Include footer text for each page (html-code)	1.2	No
bottom	Include bottom text for each page (html-code)	1.2	No
link	Create links to javadoc output at the given URL	1.2	No
linkoffline	Link to docs at <url> using package list at <url2>	1.2	No
group	Group specified packages together in overview page	1.2	No

nodeprecated	Do not include @deprecated information	all	No
nodeprecatedlist	Do not generate deprecated list	1.2	No
notree	Do not generate class hierarchy	all	No
noindex	Do not generate index	all	No
nohelp	Do not generate help link	1.2	No
nonavbar	Do not generate navigation bar	1.2	No
serialwarn	FUTURE: Generate warning about @serial tag	1.2	No
helpfile	FUTURE: Specifies the HTML help file to use	1.2	No
stylesheetfile	Specifies the CSS stylesheet to use	1.2	No
charset	FUTURE: Charset for cross-platform viewing of generated documentation	1.2	No
docencoding	Output file encoding name	1.1	No
doclet	Specifies the class file that starts the doclet used in generating the documentation.	1.2	No
docletpath	Specifies the path to the doclet class file that is specified with the -doclet option.	1.2	No
docletpathref	Specifies the path to the doclet class file that is specified with the -doclet option by reference [p 14] to a PATH defined elsewhere.	1.2	No
additionalparam	Lets you add additional parameters to the javadoc command line. Useful for doclets. Parameters containing spaces need to be quoted using ".	1.2	No
failonerror	Stop the buildprocess if the command exits with a returncode other than 0.	all	No

5.26.3 Parameters specified as nested elements

5.26.3.1 link

Create link to javadoc output at the given URL. This performs the same role as the link and linkoffline attributes. You can use either syntax (or both at once), but with the nested elements you can easily specify multiple occurrences of the arguments.

5.26.3.2 Parameters

Attribute	Description	Required
href	The URL for the external documentation you wish to link to	Yes
offline	True if this link is not available online at the time of generating the documentation	No
packagelistLoc	The location to the directory containing the package-list file for the external documentation	Only if the offline attribute is true

5.26.3.3 groups

Separates packages on the overview page into whatever groups you specify, one group per table. This performs the same role as the group attribute. You can use either syntax (or both at once), but with the nested elements you can easily specify multiple occurrences of the arguments.

Attribute	Description	Required
title	Title of the group	Yes
packages	List of packages to include in that group	Yes

5.26.3.4 doclet

The doclet nested element is used to specify the doclet that javadoc will use to process the input source files. A number of the standard javadoc arguments are actually arguments of the standard doclet. If these are specified in the javadoc task's attributes, they will be passed to the doclet specified in the `<doclet>` nested element. Such attributes should only be specified, therefore, if they can be interpreted by the doclet in use.

If the doclet requires additional parameters, these can be specified with `<param>` elements within the `<doclet>` element. These paramaters are restricted to simple strings. An example usage of the doclet element is shown below:

```
<javadoc ...>
   <doclet name="theDoclet"
           path="path/to/theDoclet">
      <param name="-foo" value="foovalue"/>
      <param name="-bar" value="barvalue"/>
   </doclet>
</javadoc>
```

5.26.3.5 sourcepath, classpath and bootclasspath

Javadoc's *sourcepath*, *classpath* and *bootclasspath* attributes are PATH like structure [p 12] and can also be set via nested *sourcepath*, *classpath* and *bootclasspath* elements respectively.

5.26.4 Example

```
<javadoc packagenames="com.dummy.test.*"
         sourcepath="src"
         destdir="docs/api"
         author="true"
         version="true"
         use="true"
         windowtitle="Test API"
         doctitle="<h1>Test</h1>"
         bottom="<i>Copyright &#169; 2000 Dummy Corp. All Rights Reserved.</i>">
  <group title="Group 1 Packages" packages="com.dummy.test.a*"/>
  <group title="Group 2 Packages" packages="com.dummy.test.b*"/>
  <link offline="true" href="http://java.sun.com/products/jdk/1.2/docs/api/" packagelistLoc="C:\tmp"/>
  <link href="http://developer.java.sun.com/developer/products/xml/docs/api/"/>
lt;/javadoc>
```

5.27 Mail

5.27.1 Description

A task to send SMTP email.

5.27.2 Parameters

Attribute	Description	Required
from	Email address of sender.	Yes
tolist	Comma-separated list of recipients.	Yes
message	Message to send in the body of the email.	Yes
files	Filename(s) of text to send in the body of the email. Multiple files are comma-separated.	
mailhost	Host name of the mail server.	No, default to "local-host"
subject	Email subject line.	No

5.27.3 Examples

```
<mail from="me" tolist="you" subject="Results of nightly build"
      files="build.log"/>
```

Sends an eMail from *me* to *you* with a subject of *Results of nightly build* and includes the contents of *build.log* in the body of the message.

5.28 Mkdir

5.28.1 Description

Creates a directory. Also non-existent parent directories are created, when necessary.

5.28.2 Parameters

Attribute	Description	Required
dir	the directory to create.	Yes

5.28.3 Examples

```
<mkdir dir="${dist}"/>
```

creates a directory `${dist}`.

```
<mkdir dir="${dist}/lib"/>
```

creates a directory `${dist}/lib`.

5.29 Move

5.29.1 Description

Moves a file to a new file or directory, or sets of files to a new directory. By default, the destination file is overwritten if it already exists. When *overwrite* is turned off, then files are only moved if the source file is newer than the destination file, or when the destination file does not exist.

FileSet [p 20] s are used to select sets of files to move to the *todir* directory.

5.29.2 Parameters

Attribute	Description	Required
file	the file to move	One of *file* or at least one nested fileset element
tofile	the file to move to	With the *file* attribute, either *tofile* or *todir* can be used. With a nested fileset, only *todir* is allowed.
todir	the directory to move to	
overwrite	overwrite existing files even if the destination files are newer (default is "true")	No
filtering	indicates whether token filtering should take place during the move. See the filter [p 50] task for a description of how filters work.	No
flatten	ignore directory structure of source directory, copy all files into a single directory, specified by the *todir* attribute (default is "false").Note that you can achieve the same effect by using a flatten mapper [p 22]	No
includeEmptyDirs	Copy empty directories included with the nested FileSet(s). Defaults to "yes".	No

5.29.3 Parameters specified as nested elements

5.29.3.1 mapper

You can define file name transformations by using a nested mapper [p 21] element. The default mapper used by `<copy>` is the identity [p 21] .

5.29.4 Examples

Move a single file (rename a file)

```
<move file="file.orig" tofile="file.moved"/>
```

Move a single file to a directory

```
<move file="file.orig" todir="dir/to/move/to"/>
```

Move a directory to a new directory

```
<move todir="new/dir/to/move/to">
  <fileset dir="src/dir"/>
</move>
```

Move a set of files to a new directory

```
<move todir="some/new/dir" >
  <fileset dir="my/src/dir" >
    <include name="**/*.jar"/>
    <exclude name="**/ant.jar"/>
  </fileset>
</move>
```

Append ".bak" to the names of all files in a directory.

```
<move todir="my/src/dir" >
  <fileset dir="my/src/dir" >
    <exclude name="**/*.bak"/>
  </fileset>
  <mapper type="glob" from="*" to="*.bak"/>
</move>
```

5.30 Patch

5.30.1 Description

Applies a diff file to originals.

5.30.2 Parameters

Attribute	Description	Required
patchfile	the file that includes the diff output	Yes
originalfile	the file to patch	No, tries to guess it from the diff file
backups	Keep backups of the unpatched files	No
quiet	Work silently unless an error occurs	No
reverse	Assume patch was created with old and new files swapped.	No
ignorewhitespace	Ignore whitespace differences.	No
strip	Strip the smallest prefix containing *num* leading slashes from filenames.	No

5.30.3 Examples

```
<patch patchfile="module.1.0-1.1.patch"/>
```

applies the diff included in *module.1.0-1.1.patch* to the files in base directory guessing the filename(s) from the diff output.

```
<patch patchfile="module.1.0-1.1.patch" strip="1"/>
```

like above but one leading directory part will be removed. i.e. if the diff output looked like

```
--- a/mod1.0/A  Mon Jun  5 17:28:41 2000
+++ a/mod1.1/A  Mon Jun  5 17:28:49 2000
```

the leading *a/* will be stripped.

5.31 Property

5.31.1 Description

Sets a property (by name and value), or set of properties (from file or resource) in the project.

When a property was set by the user, or was a property in a parent project (that started this project with the ant task [p 25]), then this property cannot be set, and will be ignored. This means that properties set outside the current project always override the properties of the current project.

There are five ways to set properties:

- By supplying both the *name* and *value* attribute.
- By supplying both the *name* and *refid* attribute.
- By setting the *file* attribute with the filename of the property file to load. This property file has the format as defined by the file used in the class java.util.Properties.
- By setting the *resource* attribute with the resource name of the property file to load. This property file has the format as defined by the file used in the class java.util.Properties.
- By setting the *environment* attribute with a prefix to use. Properties will be defined for every environment variable by prefixing the supplied name and a period to the name of the variable.

Although combinations of the three ways are possible, only one should be used at a time. Problems might occur with the order in which properties are set, for instance.

The value part of the properties being set, might contain references to other properties. These references are resolved at the time these properties are set. This also holds for properties loaded from a property file.

5.31.2 Parameters

Attribute	Description	Required
name	the name of the property to set.	Yes
value	the value of the property.	Yes
refid	Reference [p 14] to an object defined elsewhere. Only yields reasonable results for references to PATH like structures [p 12] or properties.	
resource	the resource name of the property file.	
file	the filename of the property file .	
location	Sets the property to the absolute filename of the given file. If the value of this attribute is an absolute path, it is left unchanged (with / and \ characters converted to the current platforms conventions). Otherwise it is taken as a path relative to the project's basedir and expanded.	
environment	the prefix to use when retrieving environment variables. Thus if you specify environment="myenv" you will be able to access OS-specific environment variables via property names "myenv.PATH" or "myenv.TERM". Note that if you supply a property name with a final "." it will not be doubled. ie environment="myenv." will still allow access of environment variables through "myenv.PATH" and "myenv.TERM". This functionality is currently only implemented on select platforms. Feel free to send patches to increase the number of platforms this functionality is supported on ;)	
classpath	the classpath to use when looking up a resource.	No
classpathref	the classpath to use when looking up a resource, given as reference [p 14] to a PATH defined elsewhere..	No

5.31.3 Parameters specified as nested elements

5.31.3.1 classpath

`Property`'s *classpath* attribute is a PATH like structure [p 12] and can also be set via a nested *classpath* element.

5.31.4 Examples

```
<property name="foo.dist" value="dist"/>
```

sets the property `foo.dist` to the value "dist".

```
<property file="foo.properties"/>
```

reads a set of properties from a file called "foo.properties".

```
<property resource="foo.properties"/>
```

reads a set of properties from a resource called "foo.properties".

Note that you can reference a global properties file for all of your Ant builds using the following:

```
<property file="${user.home}/.ant-global.properties"/>
```

since the "user.home" property is defined by the Java virtual machine to be your home directory. This technique is more appropriate for Unix than Windows since the notion of a home directory doesn’t exist on Windows. On the JVM that I tested, the home directory on Windows is "C:\". Different JVM implementations may use other values for the home directory on Windows.

5.32 Rename

5.32.1 Deprecated

This task has been deprecated. Use the Move task instead.

5.32.2 Description

Renames a given file.

5.32.3 Parameters

Attribute	Description	Required
src	file to rename.	Yes
dest	new name of the file.	Yes
replace	Enable replacing of existing file (default: on).	No

5.32.4 Examples

```
<rename src="foo.jar" dest="${name}-${version}.jar"/>
```

Renames the file `foo.jar` to `${name}-${version}.jar` (assuming `name` and `version` being predefined properties). If a file named `${name}-${version}.jar` already exists, it will be removed prior to renaming `foo.jar`.

5.33 Replace

5.33.1 Description

Replace is a directory based task for replacing the occurrence of a given string with another string in selected file.

If you want to replace a text that crosses line boundaries, you must use a nested `<replacetoken>` element.

5.33.2 Parameters

Attribute	Description	Required
file	file for which the token should be replaced.	Exactly one of the two.
dir	The base directory to use when replacing a token in multiple files.	
token	the token which must be replaced.	Yes, unless a nested `replace-token` element is used.
value	the new value for the token. When omitted, an empty string ("") is used.	No
propertyFile	valid property file from which properties specified using nested `<replacefilter>` elements are drawn.	Yes only if *property* attribute of `<replacefilter>` is used.
includes	comma separated list of patterns of files that must be included. All files are included when omitted.	No
includesfile	the name of a file. Each line of this file is taken to be an include pattern	No
excludes	comma separated list of patterns of files that must be excluded. No files (except default excludes) are excluded when omitted.	No
excludesfile	the name of a file. Each line of this file is taken to be an exclude pattern	No
defaultexcludes	indicates whether default excludes should be used or not ("yes"/"no"). Default excludes are used when omitted.	No

5.33.3 Examples

```
<replace file="${src}/index.html" token="@@@" value="wombat"/>
```

replaces occurrences of the string "@@@" with the string "wombat", in the file `${src}/index.html`.

5.33.4 Parameters specified as nested elements

This task forms an implicit FileSet [p 20] and supports all attributes of `<fileset>` as well as the nested `<include>`, `<exclude>` and `<patternset>` elements.

If either the text you want to replace or the replacement text cross line boundaries, you can use nested elements to specify them.

5.33.5 Examples

```
<replace dir="${src}" value="wombat">
  <include name="**/*.html"/>
  <replacetoken><![CDATA[multi line
token]]></replacetoken>
</replace>
```

replaces occurrences of the string "multi line*n*token" with the string "wombat", in all HTML files in the directory `${src}`.Where *\n* is the platform specific line separator.

```
<replace file="${src}/index.html">
  <replacetoken><![CDATA[two line
token]]></replacetoken>
  <replacevalue><![CDATA[two line
token]]></replacevalue>
</replace>
```

5.33.5.1 replacefilter

In addition to allowing for multiple replacements, optional nested `<replacefilter>` elements allow replacement values to be extracted from a property file. The name of this file is specified using the `<replace>` attribute *propertyFile*.

Attribute	Description	Required
token	The string to search for.	Yes
value	The replacement string.	Either may be specified, but not both. Both can be omitted, if desired.
property	Name of the property whose value is to serve as the replacement value.	

If neither *value* nor *property* is used, the value provided using the `<replace>` attribute *value* and/or the `<replacevalue>` element is used. If no value was specified using either of these options, the token is replaced with an empty string.

5.33.6 Examples

```
<replace
    file="configure.sh"
    value="defaultvalue"
    propertyFile="source/name.properties" >
  <replacefilter
    token="@token1@" />
  <replacefilter
    token="@token2@"
    value="value2"/>
  <replacefilter
    token="@token3@"
    property="property.key"/>
</replace>
```

In file `configure.sh`, replace all instances of "@token1@" with "defaultvalue", all instances of "@token2@" with "value2", and all instances of "@token3@" with the value of the property "property.key", as it appears in property file `src/name.properties`.

Note: It is possible to use either the *token*/`<replacetoken>` and *value*/`<replacevalue>` attributes/elements, the nested replacefilter elements, or both in the same operation.

5.34 Rmic

5.34.1 Description

Runs the rmic compiler for a certain class.

Rmic can be run on a single class (as specified with the classname attribute) or a number of classes at once (all classes below base that are neither _Stub nor _Skel classes).

It is possible to refine the set of files that are being rmiced. This can be done with the *includes*, *includesfile*, *excludes*, *excludesfile* and *defaultexcludes* attributes. With the *includes* or *includesfile* attribute you specify the files you want to have included by using patterns. The *exclude* or *excludesfile* attribute is used to specify the files you want to have excluded. This is also done with patterns. And finally with the *defaultexcludes* attribute, you can specify whether you want to use default exclusions or not. See the section on directory based tasks [p 16] , on how the inclusion/exclusion of files works, and how to write patterns.

This task forms an implicit FileSet [p 20] and supports all attributes of `<fileset>` (`dir` becomes `base`) as well as the nested `<include>`, `<exclude>` and `<patternset>` elements.

5.34.2 Parameters

Attribute	Description	Required
base	the location to store the compiled files.	Yes
classname	the class for which to run `rmic`.	No
filtering	indicates whether token filtering should take place	No
sourcebase	Pass the "-keepgenerated" flag to rmic and move the generated source file to the base directory.	No
stubversion	Specify the JDK version for the generated stub code. Specify "1.1" to pass the "-v1.1" option to rmic.	No
classpath	The classpath to use during compilation	No
classpathref	The classpath to use during compilation, given as reference [p 14] to a PATH defined elsewhere	No
includes	comma separated list of patterns of files that must be included. All files are included when omitted.	No
includesfile	the name of a file. Each line of this file is taken to be an include pattern	No
excludes	comma separated list of patterns of files that must be excluded. No files (except default excludes) are excluded when omitted.	No
excludesfile	the name of a file. Each line of this file is taken to be an exclude pattern	No
defaultexcludes	indicates whether default excludes should be used or not ("yes"/"no"). Default excludes are used when omitted.	No
verify	check that classes implement Remote before handing them to rmic (default is false)	No
iiop	indicates that portable (RMI/IIOP) stubs should be generated	No
iiopopts	additional arguments for IIOP class generation	No
idl	indicates that IDL output files should be generated	No
idlopts	additional arguments for IDL file generation	No
debug	generate debug info (passes -g to rmic). Defaults to false.	No

5.34.3 Parameters specified as nested elements

5.34.3.1 classpath

`Rmic`'s *classpath* attribute is a PATH like structure [p 12] and can also be set via a nested *classpath* elements.

5.34.4 Examples

```
<rmic classname="com.xyz.FooBar" base="${build}/classes"/>
```

runs the rmic compiler for the class `com.xyz.FooBar`. The compiled files will be stored in the directory `${build}/classes`.

```
<rmic base="${build}/classes" includes="**/Remote*.class"/>
```

runs the rmic compiler for all classes with `.class` files below `${build}/classes` whose classname starts with *Remote*. The compiled files will be stored in the directory `${build}/classes`.

5.35 SignJar

5.35.1 Description

Signs a jar or zip file with the javasign command line tool.

5.35.2 Parameters

Attribute	Description	Required
jar	the jar file to sign	Yes.
alias	the alias to sign under	Yes.
storepass	password for keystore integrity.	Yes.
keystore	keystore location	No
storetype	keystore type	No
keypass	password for private key (if different)	No
sigfile	name of .SF/.DSA file	No
signedjar	name of signed JAR file	No
verbose	(true | false) verbose output when signing	No
internalsf	(true | false) include the .SF file inside the signature block	No
sectionsonly	(true | false) don’t compute hash of entire manifest	No

5.35.3 Examples

```
<signjar jar="${dist}/lib/ant.jar" alias="apache-group"
storepass="secret"/>
```

signs the ant.jar with alias "apache-group" accessing the keystore and private key via "secret" password.

5.36 Sql

5.36.1 Description

Executes a series of sql statement via JDBC to a database. Statements can either be read in from a text file using the src attribute or from between the enclosing sql tags.

Multiple statements can be set and each statement is delimited from the next use a semi-colon. Individual lines within the statements can be commented using either -- or // at the start of the line.

The auto-commit attribute specifies whether auto commit should be turned on or off whilst executing the statements. If auto-commit is turned on each statement will be executed and committed. If it is turned off the statements will all be executed as one transaction.

The *onerror* attribute specifies how to proceed when an error occurs during the execution of one of the statements. The possible values are: **continue** execution, only show the error; **stop** execution and commit transaction; and **abort** execution and transaction and fail task.

5.36.2 Parameters

Attribute	Description	Required
driver	Class name of the jdbc driver	Yes
url	Database connection url	Yes
userid	Database user name	Yes
password	Database password	Yes
src	File containing sql statements	Yes, unless statements enclosed within tags
autocommit	Auto commit flag for database connection (default false)	No, default "false"
print	Print result sets from the statements (default false)	No, default "false"
showheaders	Print headers for result sets from the statements (default true)	No, default "true"
output	Output file for result sets (defaults to System.out)	No (print to System.out by default)
classpath	Classpath used to load driver	No (use system classpath)
onerror	Action to perform when statement fails: continue, stop, abort	No, default "abort"
rdbms	Execute task only if this rdbms	No (no restriction)
version	Execute task only if rdbms version match	No (no restriction)

5.36.3 Parameters specified as nested elements

5.36.3.1 transaction

Use nested `<transaction>` elements to specify multiple blocks of commands to the executed executed in the same connection but different transactions. This is particularly useful when there are multiple files to execute on the same schema.

Attribute	Description	Required
src	File containing sql statements	Yes, unless statements enclosed within tags

5.36.3.2 classpath

`Sql`'s *classpath* attribute is a PATH like structure [p 12] and can also be set via a nested *classpath* element. It is used to load the JDBC classes.

The

5.36.4 Examples

```
<sql
    driver="org.database.jdbcDriver"
    url="jdbc:database-url"
    userid="sa"
    password="pass"
    src="data.sql"
/>
```

Connects to the database given in url as the sa user using the org.database.jdbcDriver and executes the sql statements contained within the file data.sql

```
<sql
    driver="org.database.jdbcDriver"
    url="jdbc:database-url"
    userid="sa"
    password="pass"
    >
insert
into table some_table
values(1,2,3,4);

truncate table some_other_table;
</sql>
```

Connects to the database given in url as the sa user using the org.database.jdbcDriver and executes the two sql statements inserting data into some_table and truncating some_other_table

Note that you may want to enclose your statements in <![CDATA[...]]> sections so you don't need to escape <, > & or other special characters. For example:

```
<sql
    driver="org.database.jdbcDriver"
    url="jdbc:database-url"
    userid="sa"
    password="pass"
    ><![CDATA[

update some_table set column1 = column1 + 1 where column2 < 42;

]]></sql>
```

The following connects to the database given in url as the sa user using the org.database.jdbcDriver and executes the sql statements contained within the files data1.sql, data2.sql and data3.sql and then executes the truncate operation on *some_other_table*.

```
<sql
    driver="org.database.jdbcDriver"
    url="jdbc:database-url"
    userid="sa"
    password="pass" >
  <transaction  src="data1.sql" />
  <transaction  src="data2.sql" />
  <transaction  src="data3.sql" />
  <transaction>
    truncate table some_other_table;
  </transaction>
</sql>
```

The following connects to the database given in url as the sa user using the org.database.jdbcDriver and executes the sql statements contained within the file data.sql, with output piped to outputfile.txt, searching /some/jdbc.jar as well as the system classpath for the driver class.

```
<sql
    driver="org.database.jdbcDriver"
    url="jdbc:database-url"
    userid="sa"
    password="pass"
    src="data.sql"
    print="yes"
    output="outputfile.txt"
    >
<classpath>
        <pathelement location="/some/jdbc.jar">
</classpath>
</sql>
```

The following will only execute if the RDBMS is "oracle" and the version starts with "8.1."

```
<sql
    driver="org.database.jdbcDriver"
    url="jdbc:database-url"
    userid="sa"
```

```
    password="pass"
    src="data.sql"
    rdbms="oracle"
    version="8.1."
    >
insert
into table some_table
values(1,2,3,4);

truncate table some_other_table;
</sql>
```

5.37 Style

5.37.1 Description

Process a set of documents via XSLT.

This is useful for building views of XML based documentation, or in generating code.

It is possible to refine the set of files that are being copied. This can be done with the *includes*, *includesfile*, *excludes*, *excludesfile* and *defaultexcludes* attributes. With the *includes* or *includesfile* attribute you specify the files you want to have included by using patterns. The *exclude* or *excludesfile* attribute is used to specify the files you want to have excluded. This is also done with patterns. And finally with the *defaultexcludes* attribute, you can specify whether you want to use default exclusions or not. See the section on directory based tasks [p 16] , on how the inclusion/exclusion of files works, and how to write patterns.

This task forms an implicit FileSet [p 20] and supports all attributes of `<fileset>` (`dir` becomes `basedir`) as well as the nested `<include>`, `<exclude>` and `<patternset>` elements.

Style supports the use of a <param> element which is use to pass values to an <xsl:param> declaration.

5.37.2 Parameters

Attribute	Description	Required
basedir	where to find the source XML file.	Yes
destdir	directory where to store the results.	Yes
extension	desired file extension to be used for the targets. If not specified, the default is "html".	No
style	name of the stylesheet to use.	Yes
processor	name of the XSLT processor to use. Permissible values are "trax" for a TraX compliant processor, "xslp" for the XSL:P processor, "xalan" for the Apache XML Xalan (version 1) processor, or the name of an arbitrary XSLTLiaison class. Defaults to trax, followed by xslp then xalan (in that order). The first one found in your class path is the one that is used.	No
includes	comma separated list of patterns of files that must be included. All files are included when omitted.	No
includesfile	the name of a file. Each line of this file is taken to be an include pattern	No
excludes	comma separated list of patterns of files that must be excluded. No files (except default excludes) are excluded when omitted.	No
excludesfile	the name of a file. Each line of this file is taken to be an exclude pattern	No
defaultexcludes	indicates whether default excludes should be used or not ("yes"/"no"). Default excludes are used when omitted.	No
in	specifies a single XML document to be styled. Should be used with the out attribute.	No
out	specifies the output name for the styled result from the in attribute.	No

5.37.3 Parameters specified as nested elements

5.37.3.1 param

Param is used to pass a parameter to the XSL stylesheet.

5.37.3.2 Parameters

Attribute	Description	Required
name	Name of the XSL parameter	Yes
expression	XSL expression to be placed into the param. To pass a text value into the style sheet it needs to be escaped using single quotes.	Yes

5.37.4 Examples

```
<style basedir="doc" destdir="build/doc"
       extension="html" style="style/apache.xsl"/>
```

5.37.4.1 Using XSL parameters

```
<style basedir="doc" destdir="build/doc"
               extension="html" style="style/apache.xsl">
        <param name="date" expression="'07-01-2000'"/>
</style>
```

This will replace an xsl:param definition<xsl:param name="date"></xsl:param> with the text value 07-01-2000

5.38 Tar

5.38.1 Description

Creates a tar archive.

The *basedir* attribute is the reference directory from where to tar.

This task is a directory based task [p 16] and, as such, forms an implicit Fileset [p 20] . This defines which files, relative to the *basedir*, will be included in the archive. The tar task supports all the attributes of Fileset to refine the set of files to be included in the implicit fileset.

In addition to the implicit fileset, the tar task supports nested filesets. These filesets are extended to allow control over the access mode, username and groupname to be applied to the tar entries. This is useful, for example, when preparing archives for Unix systems where some files need to have execute permission.

Early versions of tar did not support path lengths greater than 100 characters. Modern versions of tar do so, but in incompatible ways. The behaviour of the tar task when it encounters such paths is controlled by the *longfile* attribute. If the longfile attribute is set to `fail`, any long paths will cause the tar task to fail. If the longfile attribute is set to `truncate`, any long paths will be truncated to the 100 character maximum length prior to adding to the archive. If the value of the longfile attribute is set to `omit` then files containing long paths will be omitted from the archive. Either option ensures that the archive can be untarred by any compliant version of tar. If the loss of path or file information is not acceptable, and it rarely is, longfile may be set to the value `gnu`. The tar task will then produce a GNU tar file which can have arbitrary length paths. Note however, that the resulting archive will only be able to be untarred with GNU tar. The default for the longfile attribute is `warn` which behaves just like the gnu option except that it produces a warning for each file path encountered that does not match the limit.

Note that this task does not perform compression. You might want to use the GZip [p 57] task to prepare a .tar.gz package.

5.38.2 Parameters

Attribute	Description	Required
tarfile	the tar-file to create.	Yes
basedir	the directory from which to zip the files.	No
longfile	Determines how long files (>100 chars) are to be handled. Allowable values are "truncate", "fail", "warn", "omil" and "gnu". Default is "warn".	No
includes	comma separated list of patterns of files that must be included. All files are included when omitted.	No
includesfile	the name of a file. Each line of this file is taken to be an include pattern	No
excludes	comma separated list of patterns of files that must be excluded. No files (except default excludes) are excluded when omitted.	No
excludesfile	the name of a file. Each line of this file is taken to be an exclude pattern	No
defaultexcludes	indicates whether default excludes should be used or not ("yes"/"no"). Default excludes are used when omitted.	No

5.38.3 Nested Elements

The tar task supports nested tarfileset [p 20] elements. These are extended Filesets which, in addition to the standard fileset elements, support three additional attributes

Attribute	Description	Required
mode	An 3 digit octal string, specify the user, group and other modes in the standard Unix fashion	No
username	The username for the tar entry. This is not the same as the UID, which is not currently set by the tar task.	No
groupname	The username for the tar entry. This is not the same as the GID, which is not currently set by the tar task.	No

5.38.4 Examples

```
<tar tarfile="${dist}/manual.tar" basedir="htdocs/manual"/>
<gzip zipfile="${dist}/manual.tar.gz" src="${dist}/manual.tar"/>
```

tars all files in the `htdocs/manual` directory into a file called `manual.tar` in the `${dist}` directory, then applies the gzip task to compress it.

```
<tar tarfile="${dist}/manual.tar"
     basedir="htdocs/manual"
     excludes="mydocs/**, **/todo.html"
/>
```

tars all files in the `htdocs/manual` directory into a file called `manual.tar` in the `${dist}` directory. Files in the directory `mydocs`, or files with the name `todo.html` are excluded.

```
<tar longfile="gnu"
     tarfile="${dist.base}/${dist.name}-src.tar" >
  <tarfileset dir="${dist.name}/.." mode="755" username="ant" group="ant">
    <include name="${dist.name}/bootstrap.sh"/>
    <include name="${dist.name}/build.sh"/>
  </fileset>
  <tarfileset dir="${dist.name}/.." username="ant" group="ant">
    <include name="${dist.name}/**"/>
    <exclude name="${dist.name}/bootstrap.sh"/>
    <exclude name="${dist.name}/build.sh"/>
  </fileset>
</tar>
```

This example shows building a tar which uses the GNU extensions for long paths and where some files need to be marked as executable (mode 755) and the rest are use the default mode (read-write by owner). The first fileset selects just the executable files. The second fileset must exclude the executable files and include all others.

Note: The tar task does not ensure that a file is only selected by one fileset. If the same file is selected by more than one fileset, it will be included in the tar file twice, with the same path.

5.39 Taskdef

5.39.1 Description

Adds a task definition to the current project, such that this new task can be used in the current project. Two attributes are needed, the name that identifies this task uniquely, and the full name of the class (including the packages) that implements this task.

Taskdef should be used to add your own tasks to the system. See also "Writing your own task [p 195] ".

5.39.2 Parameters

Attribute	Description	Required
name	the name of the task	Yes
classname	the full class name implementing the task	Yes
classpath	the classpath to use when looking up `classname`.	No

5.39.3 Parameters specified as nested elements

5.39.3.1 classpath

`Taskdef`'s *classpath* attribute is a PATH like structure [p 12] and can also be set via a nested *classpath* element.

5.39.4 Examples

```
<taskdef name="myjavadoc" classname="com.mydomain.JavadocTask"/>
```

makes a task called `myjavadoc` available to Ant. The class `com.mydomain.JavadocTask` implements the task.

5.40 Touch

5.40.1 Description

Changes the modification time of a file and possibly creates it at the same time.

For JDK 1.1 only the creation of new files with a modification time of now works, all other cases will emit a warning.

5.40.2 Parameters

Attribute	Description	Required
file	the name of the file	Yes
millis	specifies the new modification time of the file in milliseconds since midnight Jan 1 1970	No
datetime	specifies the new modification time of the file in the format MM/DD/YYYY HH:MM AM_or_PM.	No

If both `millis` and `datetime` are omitted the current time is assumed.

5.40.3 Examples

```
<touch file="myfile"/>
```

creates `myfile` if it doesn't exist and changes the modification time to the current time.

```
<touch file="myfile" datetime="06/28/2000 2:02 pm"/>
```

creates `myfile` if it doesn't exist and changes the modification time to Jun, 28 2000 2:02 pm (14:02 for those used to 24 hour times).

5.41 Tstamp

5.41.1 Description

Sets the DSTAMP, TSTAMP and TODAY properties in the current project. The DSTAMP is in the "yyyymmdd" format, the TSTAMP is in the "hhmm" format and TODAY is "month day year".

These properties can be used in the buildfile, for instance, to create timestamped filenames or used to replace placeholder tags inside documents to indicate, for example, the release date. The best place for this task is in your initialization target.

5.41.2 Parameters

Attribute	Description	Required
No parameters		

5.41.3 Nested Elements

The tstamp task supports a format nested element which allows a property to be given the current date and time in a given format. The date/time patterns are as defined in the Java SimpleDateFormat class.

Attribute	Description	Required
property	The property which is to receive the date/time string in the given pattern	Yes
pattern	The date/time pattern to be used. The values are defined by the Java SimpleDateFormat class	Yes

5.41.4 Examples

Set the standard DSTAMP, TSTAMP and TODAY properties according to the formats above

```
<tstamp/>
```

As for the above example, set the standard properties and also set the property "TODAY_UK" with the date/time pattern "d MMM yyyy"

```
<tstamp>
  <format property="TODAY_UK" pattern="d MMMM yyyy">
</tstamp>
```

5.42 Unjar/Unwar/Unzip

5.42.1 Description

Unzips a zip-, war- or jarfile.

For JDK 1.1 "last modified time" field is set to current time instead of being carried from zipfile.

File permissions will not be restored on extracted files.

5.42.2 Parameters

Attribute	Description	Required
src	zipfile to expand.	Yes
dest	directory where to store the expanded files.	Yes

5.42.3 Examples

```
<unzip src="${tomcat_src}/tools-src.zip" dest="${tools.home}" />
```

5.43 Untar

5.43.1 Description

Untars a tarfile.

Note: File permissions will not be restored on extracted files.

For JDK 1.1 "last modified time" field is set to current time instead of being carried from tarfile.

The untar task recognizes the long pathname entries used by GNU tar.

5.43.2 Parameters

Attribute	Description	Required
src	tarfile to expand.	Yes
dest	directory where to store the expanded files.	Yes

5.43.3 Examples

```
<gunzip src="tools.tar.gz"/>
<untar src="tools.tar" dest="${tools.home}"/>
```

5.44 Unjar/Unwar/Unzip

5.44.1 Description

Unzips a zip-, war- or jarfile.

For JDK 1.1 "last modified time" field is set to current time instead of being carried from zipfile.

File permissions will not be restored on extracted files.

5.44.2 Parameters

Attribute	Description	Required
src	zipfile to expand.	Yes
dest	directory where to store the expanded files.	Yes

5.44.3 Examples

```
<unzip src="${tomcat_src}/tools-src.zip" dest="${tools.home}" />
```

5.45 Unjar/Unwar/Unzip

5.45.1 Description

Unzips a zip-, war- or jarfile.

For JDK 1.1 "last modified time" field is set to current time instead of being carried from zipfile.

File permissions will not be restored on extracted files.

5.45.2 Parameters

Attribute	Description	Required
src	zipfile to expand.	Yes
dest	directory where to store the expanded files.	Yes

5.45.3 Examples

```
<unzip src="${tomcat_src}/tools-src.zip" dest="${tools.home}" />
```

5.46 Uptodate

5.46.1 Description

Sets a property if a target files are more up to date than a set of Source files. Source files are specified by nested <srcfiles> elements, these are FileSet [p 20] s, while target files are specified using a nested mapper [p 21] element.

The value part of the property being set is *true* if the timestamp of the target files is more recent than the timestamp of every corresponding source file.

The default behavior is to use a merge mapper [p 22] with the `to` attribute set to the value of the targetfile attribute.

Normally, this task is used to set properties that are useful to avoid target execution depending on the relative age of the specified files.

5.46.2 Parameters

Attribute	Description	Required
property	the name of the property to set.	Yes
targetfile	the file for which we want to determine the status.	Yes, unless a nested mapper element is present.

5.46.3 Examples

```
<uptodate property="xmlBuild.notRequired" targetfile="${deploy}\xmlClasses.jar" >
  <srcfiles dir= "${src}/xml" includes="**/*.dtd"/>
</uptodate>
```

sets the property *`xmlBuild.notRequired`* to the value "true" if the *${deploy}/xmlClasses.jar* is more up to date than any of the DTD files in the *${src}/xml* directory.

This can be written as

```
<uptodate property="xmlBuild.notRequired"  >
  <srcfiles dir= "${src}/xml" includes="**/*.dtd"/>
  <mapper type="merge" to="${deploy}\xmlClasses.jar"/>
</uptodate>
```

as well.

5.47 War

5.47.1 Description

An extension of the Jar [p 58] task with special treatment for files that should end up in the `WEB-INF/lib`, `WEB-INF/classes` or `WEB-INF` directories of the Web Application Archive.

(The War task is a shortcut for specifying the particular layout of a WAR file. The same thing can be accomplished by using the *prefix* and *fullpath* attributes of zipfilesets in a Zip or Jar task.)

The extended zipfileset element from the zip task (with attributes *prefix*, *fullpath*, and *src*) is available in the War task.

5.47.2 Parameters

Attribute	Description	Required
warfile	the war-file to create.	Yes
webxml	The deployment descriptor to use (WEB-INF/web.xml).	Yes
basedir	the directory from which to jar the files.	No
compress	Not only store data but also compress them, defaults to true	No
includes	comma separated list of patterns of files that must be included. All files are included when omitted.	No
includesfile	the name of a file. Each line of this file is taken to be an include pattern	No
excludes	comma separated list of patterns of files that must be excluded. No files (except default excludes) are excluded when omitted.	No
excludesfile	the name of a file. Each line of this file is taken to be an exclude pattern	No
defaultexcludes	indicates whether default excludes should be used or not ("yes"/"no"). Default excludes are used when omitted.	No
manifest	the manifest file to use.	No
whenempty	Behavior to use if no files match.	No

5.47.3 Nested elements

5.47.3.1 lib

The nested `lib` element specifies a FileSet [p 20] . All files included in this fileset will end up in the `WEB-INF/lib` directory of the war file.

5.47.3.2 classes

The nested `classes` element specifies a FileSet [p 20] . All files included in this fileset will end up in the `WEB-INF/classes` directory of the war file.

5.47.3.3 webinf

The nested `webinf` element specifies a FileSet [p 20] . All files included in this fileset will end up in the `WEB-INF` directory of the war file. If this fileset includes a file named `web.xml`, the file is ignored and you will get a warning.

5.47.4 Examples

Assume the following structure in the project's base directory:

```
thirdparty/libs/jdbc1.jar
thirdparty/libs/jdbc2.jar
build/main/com/myco/myapp/Servlet.class
src/metadata/myapp.xml
src/html/myapp/index.html
src/jsp/myapp/front.jsp
src/graphics/images/gifs/small/logo.gif
src/graphics/images/gifs/large/logo.gif
```

then the war file `myapp.war` created with

```
<war warfile="myapp.war" webxml="src/metadata/myapp.xml">
  <fileset dir="src/html/myapp"/>
  <fileset dir="src/jsp/myapp"/>
  <lib dir="thirdparty/libs">
    <exclude name="jdbc1.jar"/>
  </lib>
  <classes dir="build/main"/>
  <zipfileset dir="src/graphics/images/gifs"
              prefix="images"/>
</war>
```

will consist of

```
WEB-INF/web.xml
WEB-INF/lib/jdbc2.jar
WEB-INF/classes/com/myco/myapp/Servlet.class
META-INF/MANIFEST.MF
index.html
front.jsp
images/small/logo.gif
images/large/logo.gif
```

using Ant's default manifest file. The content of `WEB-INF/web.xml` is identical to `src/metadata/myapp.xml`.

5.48 Zip

5.48.1 Description

Creates a zipfile.

The *basedir* attribute is the reference directory from where to zip.

Note that file permissions will not be stored in the resulting zipfile.

It is possible to refine the set of files that are being zipped. This can be done with the *includes*, *includesfile*, *excludes*, *excludesfile* and *defaultexcludes* attributes. With the *includes* or *includesfile* attribute you specify the files you want to have included by using patterns. The *exclude* or *excludesfile* attribute is used to specify the files you want to have excluded. This is also done with patterns. And finally with the *defaultexcludes* attribute, you can specify whether you want to use default exclusions or not. See the section on directory based tasks [p 16] , on how the inclusion/exclusion of files works, and how to write patterns.

This task forms an implicit FileSet [p 20] and supports all attributes of `<fileset>` (`dir` becomes `basedir`) as well as the nested `<include>`, `<exclude>` and `<patternset>` elements.

Or, you may place within it nested file sets, or references to file sets. In this case `basedir` is optional; the implicit file set is *only used* if `basedir` is set. You may use any mixture of the implicit file set (with `basedir` set, and optional attributes like `includes` and optional subelements like `<include>`); explicit nested `<fileset>` elements so long as at least one fileset total is specified. The ZIP file will only reflect the relative paths of files *within* each fileset. The Zip task and its derivatives know a special form of a fileset named zipfileset that has additional attributes (described below).

The `whenempty` parameter controls what happens when no files match. If `skip` (the default), the ZIP is not created and a warning is issued. If `fail`, the ZIP is not created and the build is halted with an error. If `create`, an empty ZIP file (explicitly zero entries) is created, which should be recognized as such by compliant ZIP manipulation tools.

5.48.2 Parameters

Attribute	Description	Required
zipfile	the zip-file to create.	Yes
basedir	the directory from which to zip the files.	No
compress	Not only store data but also compress them, defaults to true	No
includes	comma separated list of patterns of files that must be included. All files are included when omitted.	No
includesfile	the name of a file. Each line of this file is taken to be an include pattern	No
excludes	comma separated list of patterns of files that must be excluded. No files (except default excludes) are excluded when omitted.	No
excludesfile	the name of a file. Each line of this file is taken to be an exclude pattern	No
defaultexcludes	indicates whether default excludes should be used or not ("yes"/"no"). Default excludes are used when omitted.	No
whenempty	Behavior when no files match.	No

5.48.3 Parameters specified as nested elements

5.48.3.1 fileset

The zip task supports any number of nested `<fileset>` [p 20] elements to specify the files to be included in the archive.

5.48.3.2 zipfileset

A `<zipfileset>` has three additional attributes: *prefix*, *fullpath*, and *src*. The *prefix* and *fullpath* attributes modify the location of the files when they are placed inside the archive. If the *prefix* attribute is set, all files in the fileset are prefixed with that path in the archive. If the *fullpath* attribute is set, the file described by the fileset is placed at that exact location in the archive. (The *fullpath* attribute can only be set for filesets that represent a single file. The *prefix* and *fullpath* attributes cannot both be set on the same fileset.) The *src* attribute may be used in place of the *dir* attribute to specify a zip file whose contents will be extracted and included in the archive. As with directories, include and exclude patterns may be used to specify a subset of the zip file for inclusion in the archive.

5.48.4 Examples

```
<zip zipfile="${dist}/manual.zip"
     basedir="htdocs/manual"
/>
```

zips all files in the `htdocs/manual` directory into a file called `manual.zip` in the `${dist}` directory.

```
<zip zipfile="${dist}/manual.zip"
     basedir="htdocs/manual"
     excludes="mydocs/**, **/todo.html"
/>
```

zips all files in the `htdocs/manual` directory. Files in the directory `mydocs`, or files with the name `todo.html` are excluded.

```
<zip zipfile="${dist}/manual.zip"
     basedir="htdocs/manual"
     includes="api/**/*.html"
     excludes="**/todo.html"
/>
```

zips all files in the `htdocs/manual` directory. Only html files under the directory `api` are zipped, and files with the name `todo.html` are excluded.

```
<zip zipfile="${dist}/manual.zip">
  <fileset dir="htdocs/manual"/>
  <fileset dir="." includes="ChangeLog.txt"/>
</zip>
```

zips all files in the `htdocs/manual` directory, and also adds the file `ChangeLog.txt` in the current directory. `ChangeLog.txt` will be added to the top of the ZIP file, just as if it had been located at `htdocs/manual/ChangeLog.txt`.

```
<zip zipfile="${dist}/manual.zip">
  <zipfileset dir="htdocs/manual" prefix="docs/user-guide"/>
  <zipfileset dir="." includes="ChangeLog27.txt" fullpath="docs/ChangeLog.txt"/>
  <zipfileset src="examples.zip" includes="**/*.html" prefix="docs/examples"/>
</zip>
```

zips all files in the `htdocs/manual` directory into the `docs/user-guide` directory in the archive, adds the file `ChangeLog27.txt` in the current directory as `docs/ChangeLog.txt`, and includes all the html files in `examples.zip` under `docs/examples`. The archive might end up containing the files:

```
docs/user-guide/html/index.html
docs/ChangeLog.txt
docs/examples/index.html
```

6 Optional Tasks

6.1 .NET classes for Ant

By Steve Loughran; steve_l@iseran.com

6.1.1 Vision

Make ant the definitive build tool for .NET projects, just as it is becoming for java projects

6.1.2 Detail

The emergent policy in big java projects is 'IDE for code and debug'; Ant for builds, test and deployment. While IDEs are great for an individual development task, Ant not only rebuilds everything insanely fast, it has great support for modern deployment tasks -ftp upload, EJB servers, SQL commands, all can be done from a relatively simple XML declaration of what you want done. Also in a large project (especially open source) there is no means of dictating which IDE to use. Ant gives IDE independence, cross platform portability and ease of extensibility.

Projects using .net should be able to use the same tool. This will avoid everyone being dependent on Visual Studio.net (if they have it) supplemented with makefiles and batch files. Anyone doing mixed java/dotnet development will be particulary grateful, as they can have a unified build process.

If at sometime in the future bits of .net do appear on different platforms then ant based build files will migrate -once the ant tasks have been migrated and any hard codings of DOS-descendant file system conventions removed from the build.xml files.

6.1.3 Key .net Tasks

1. csc - csharp compiler - *.csc -> output
2. ilasm - IL assembler - *.il ->.exe or .dll
3. ResX - ResXToResources.exe .resx -> .resources
4. al - assembly linker (.dll) ->
5. Signcode - .exe + keys -> .exe
6. Resgen - .properties -> .resources
7. Type Library Exporter (TlbExp.exe) .dll -> .tlb
8. TlbImp.exe Type Library Importer
9. RegAsm.exe Assembly Registration Tool .dll -> .reg | registration
10. WebServiceUtil -SOAP proxy import, SDL generation tool

6.1.4 Ant Wrapper for net tasks

Rarely used .net command line tools can be invoked manually. The key tasks to address are the common steps in a build process and those which benefit from the file pattern matching function.

The esoteric tasks can then supported on demand by those who need them. The initial .net tasks should provide a foundation for that demand creation to be simple.

6.1.5 Plan

1. code up C sharp task and distribute for feedback & identify possible aid
2. Use for a real end to end development task
3. Add the next task I need (whatever that is)
4. Refactor to produce a base class for .net tasks
5. Leave it other people write all the other tasks

6.1.6 Risks

- The .net SDK is unstable; the next builds will change the commands, arguments and behaviours. Probability: 100%. Impact: 2 days.
- Portable tasks turn out to be harder than envisaged (=reworking if/when non Win32 .net build tools appear)

6.1.7 Using Ant in .net projects

To use the net tasks one needs

- A Win2K box (or any other platform which supports the .net build tools)
- JavaVM - Java1.1 or later; Java1.3 recommended.
- Ant downloaded and the environment and path variables set up so that typing 'ant' invokes ant via a batch file.
- An XML parser for ant. The usual distributions include an appropriate version parser.jar and jaxp.jar.
- the .net tasks archive (dotnettasks.jar) included in the ant\lib directory
- The .net sdk installed with the binaries on the path. (Test: type CSC)
- A build.xml file to describe the build process

6.1.8 Change Log

6.1.8.0.1 Version 0.3

The changes here reflect Beta-1 of the dotnet SDK and experience of use in more complex projects. This build does not work with the older SDK, primarily because the automatic reference feature references libraries only found in the new SDK version.

External changes

- Recursive inclusion of .cs and .il files
- Documentation enhanced, includes examples and details of all parameters
- The csc task automatically includes the common dotnet assemblies, so there is no need to remember to refer to 'System.dll', 'System.Web.Services', etc. This feature can be disabled by setting the 'includeDefaultReferences' flag to false.
- References can also be referred to using the ReferenceFiles parameter, which is an ant path specification. The old 'references' string is still retained.
- An 'extraoptions' attribute enables the build file to include any CSC options which are not explicitly supported in the CSC task.

Internal changes

- Some minor refactoring (move common code a method)
- Application of Jedits JavaStyle task resulted in a major reshaping of the codebase and the insertion of a blank line every second line. Significant effort was required to revert some (but not all) changes.
- Removed throws clause from methods which can't throw exceptions

The test harness has been expanded to include unicode source file (the build works but the rest of the system has 'issues' with high unicode package and method names)

6.1.8.0.2 Version 0.2

First public edition, added to the ant cvs tree. Tested on the PDC build of the dotnet SDK only, and still immature. The command execution code was refactored out into a 'NetCommand' class for re-use. The Ilasm task was added at this time.

6.1.8.0.3 Version 0.1

Initial proof of concept; very rudimentary support for CSC only.

6.1.9 Task: CSC

This task compiles CSharp source into executables or modules. This task compiles CSharp source into executables or modules. The task will only work on win2K until other platforms support csc.exe or an equivalent. CSC.exe must be on the execute path too.

All parameters are optional: <csc/> should suffice to produce a debug build of all *.cs files. References to external files do require explicit enumeration, so are one of the first attributes to consider adding.

The task is a directory based task, so attributes like **includes="**/*.cs"** and **excludes="broken.cs"** can be used to control the files pulled in. By default, all *.cs files from the project folder down are included in the command. When this happens the output file -if not specified- is taken as the first file in the list, which may be somewhat hard to control. Specifying the output file with **'outfile'** seems prudent.

Attribute	Description	Example Values
additionalModules	Semicolon separated list of modules to refer to	
defaultexcludes	indicates whether default excludes should be used or not	"true"(default) or "false"
definitions	defined constants	"RELEASE;BETA1"
debug	include debug information	"true"(default) or "false"
docFile	name of file for documentation	"doc.xml"
excludes	comma separated list of patterns of files that must be excluded. No files (except default excludes) are excluded when omitted.	
excludesfile	the name of a file. Each line of this file is taken to be an exclude pattern	
extraOptions	Any extra options which aren't explicitly supported by the CSharp task	"/warnaserror+ /baseaddress:0x12840000"
failOnError	Should a failed compile halt the build?	"true"(default) or "false"
includes	comma separated list of patterns of files that must be included. All files are included when omitted.	
includeDefaultReferences	Flag which when true automatically includes the common assemblies in dotnet beta1, and tells the compiler to link in mscore.dll	"true"(default) or "false"
includesfile	the name of a file. Each line of this file is taken to be an include pattern	
incremental	Incremental build flag. Avoid till it works	"true" or "false"(default)
mainClass	name of main class for executables	"com.example.project.entrypoint"
optimize	optimisation flag	"true" or "false"(default)
outputFile	filename of output	"example.exe"
references	Semicolon separated list of dlls to refer to	"mylib.dll;nunit.dll"
referenceFiles	Ant Path descriptioon of references to include. Wildcards should work.	
srcDir	source directory (default = project directory)	"."
targetType	Type of target	"exe", "module", "winexe" or "library"
unsafe	enable unsafe code	"true" or "false"(default)

warnLevel	level of warning currently between 1 and 4 with 4 being the strictest.	"1"-"4"
win32Icon	filename of icon to include	"res/myicon.ico"

Example

```
<csc
        optimize="true"
        debug="false"
        docFile="documentation.xml"
        warnLevel="4"
        unsafe="false"
        targetType="exe"
        incremental="false"
        definitions="RELEASE"
        excludes="src/unicode_class.cs"
        mainClass = "MainApp"
        outputFile="NetApp.exe"
        />
```

6.1.10 Task: ilasm

Task to assemble .net 'Intermediate Language' files. The task will only work on win2K until other platforms support csc.exe or an equivalent. ilasm.exe must be on the execute path too.

All parameters are optional: <il/> should suffice to produce a debug build of all *.il files. The option set is roughly compatible with the CSharp class; even though the command line options are only vaguely equivalent. [The low level commands take things like /OUT=file, csc wants /out:file ... /verbose is used some places; /quiet here in ildasm... etc.] It would be nice if someone made all the command line tools consistent (and not as brittle as the java cmdline tools)

The task is a directory based task, so attributes like **includes="*.il"** and **excludes="broken.il"** can be used to control the files pulled in. Each file is built on its own, producing an appropriately named output file unless manually specified with **outfile**

Attribute	Description	Example
defaultexcludes	indicates whether default excludes should be used or not ("yes"/"no"). Default excludes are used when omitted.	
debug	include debug information	true (default)
excludes	comma separated list of patterns of files that must be excluded. No files (except default excludes) are excluded when omitted.	
excludesfile	the name of a file. Each line of this file is taken to be an exclude pattern	
extraOptions	Any extra options which aren't explicitly supported by the ilasm task	
failOnError	Should a failed compile halt the build?	"true"(default)
includes	comma separated list of patterns of files that must be included. All files are included when omitted.	
includesfile	the name of a file. Each line of this file is taken to be an include pattern	
listing	Produce a listing (off by default). Listings go to the current output stream	"on", "off" (default)
outputFile	filename of output	"example.exe"
owner	restrict disassembly by setting the 'owner' string	"secret"
resourceFile	name of resource file to include	"resources.res"
srcDir	source directory (default = project directory)	
targetType	Type of target. library means DLL is output.	"exe"(default), "library"
verbose	output progress messages	"on", "off" (default)

Example

```
<ilasm
        outputFile="app.exe"
        verbose="on"
        listing="on"
        owner="secret"
        />
```

6.2 ANTLR

6.2.1 Description

Invokes the ANTLR (http://www.antlr.org/) Translator generator on a grammar file.

To use the ANTLR task, set the *target* attribute to the name of the grammar file to process. Optionally, you can also set the *outputdirectory* to write the generated file to a specific directory. Otherwise ANTLR writes the generated files to the directory containing the grammar file.

This task only invokes ANTLR if the grammar file is newer than the generated files.

6.2.2 Parameters

Attribute	Description	Required
target	The grammar file to process.	Yes
outputdirectory	The directory to write the generated files to. If not set, the files are written to the directory containing the grammar file.	No
fork	Run ANTLR in a separate VM.	No, default is "off"
dir	The directory to invoke the VM in. (ignored if fork is disabled)	No

6.2.3 Example

```
<antlr
    target="etc/java.g"
    outputdirectory="build/src"
/>
```

This invokes ANTLR on grammar file etc/java.g, writing the generated files to build/src.

6.3 Cab

6.3.1 Description

The cab task creates Microsoft cab archive files. It is invoked similar to the jar [p 58] or zip [p 108] tasks. This task will work on Windows using the external cabarc tool (provided by Microsoft) which must be located in your executable path.

To use this task on other platforms you need to download and compile libcabinet from http://trill.cis.fordham.edu/~barbacha/cabinet_library/.

See the section on directory based tasks [p 16] , on how the inclusion/exclusion of files works, and how to write patterns.

This task forms an implicit FileSet [p 20] and supports all attributes of `<fileset>` (`dir` becomes `basedir`) as well as the nested `<include>`, `<exclude>` and `<patternset>` elements.

6.3.2 Parameters

Attribute	Description	Required
cabfile	the name of the cab file to create.	Yes
basedir	the directory to start archiving files from.	Yes
verbose	set to "yes" if you want to see the output from the cabarc tool. defaults to "no".	No
compress	set to "no" to store files without compressing. defaults to "yes".	No
options	use to set additional command-line options for the cabarc tool. should not normally be necessary.	No
includes	comma separated list of patterns of files that must be included. All files are included when omitted.	No
includesfile	the name of a file. Each line of this file is taken to be an include pattern	No
excludes	comma separated list of patterns of files that must be excluded. No files (except default excludes) are excluded when omitted.	No
excludesfile	the name of a file. Each line of this file is taken to be an exclude pattern	No
defaultexcludes	indicates whether default excludes should be used or not ("yes"/"no"). Default excludes are used when omitted.	No

6.3.3 Examples

```
<cab cabfile="${dist}/manual.cab"
     basedir="htdocs/manual"
  />
```

cabs all files in the htdocs/manual directory into a file called manual.cab in the ${dist} directory.

```
<cab cabfile="${dist}/manual.cab"
     basedir="htdocs/manual"
     excludes="mydocs/**, **/todo.html"
  />
```

cabs all files in the htdocs/manual directory into a file called manual.cab in the ${dist} directory. Files in the directory mydocs, or files with the name todo.html are excluded.

```
<cab cabfile="${dist}/manual.cab"
     basedir="htdocs/manual"
     includes="api/**/*.html"
     excludes="**/todo.html"
     verbose="yes"
  />
```

Cab all files in the htdocs/manual directory into a file called manual.cab in the ${dist} directory. Only html files under the directory api are archived, and files with the name todo.html are excluded. Output from the cabarc tool is displayed in the build output.

6.4 ClearCase Support

- CCCheckin [p 120]
- CCCheckout [p 120]
- CCUnCheckout [p 121]
- CCUpdate [p 122]

6.4.1 CCCheckin

6.4.1.1 Description

Task to perform a Checkin command to ClearCase.

6.4.1.2 Parameters

Attribute	Values	Required
viewpath	Path to the ClearCase view file or directory that the command will operate on	No
comment	Specify a comment. Only one of comment or commentfile may be used.	No
commentfile	Specify a file containing a comment. Only one of comment or commentfile may be used.	No
nowarn	Suppress warning messages	No
preservetime	Preserve the modification time	No
keepcopy	Keeps a copy of the file with a .keep extension	No
identical	Allows the file to be checked in even if it is identical to the original	No

6.4.1.3 Examples

```
<cccheckin viewpath="c:/views/viewdir/afile"
        commentfile="acomment.txt"/>
        nowarn="true"
        identical="true"
```

Does a ClearCase *checkin* on the file *c:/views/viewdir/afile*. Comment text from the file *acomment.txt* is added to ClearCase as a comment. All warning messages are suppressed. The file is checked in even if it is *identical* to the original.

6.4.2 CCCheckout

6.4.2.1 Description

Task to perform a Checkout command to ClearCase.

6.4.2.2 Parameters

Attribute	Values	Required
viewpath	Path to the ClearCase view file or directory that the command will operate on	No
reserved	Specifies whether to check out the file as reserved or not	Yes
out	Creates a writable file under a different filename	No
nodata	Checks out the file but does not create an editable file containing its data	No
branch	Specify a branch to check out the file to	No
version	Allows checkout of a version other than main latest	No
nowarn	Suppress warning messages	No
comment	Specify a comment. Only one of comment or commentfile may be used.	No
commentfile	Specify a file containing a comment. Only one of comment or commentfile may be used.	No

6.4.2.3 Examples

```
<cccheckout viewpath="c:/views/viewdir/afile"
        reserved="true"
        branch="abranch"
        nowarn="true"
        comment="Some comment text"/>
```

Does a ClearCase *checkout* on the file *c:/views/viewdir/afile*. It is checked out as *reserved* on branch called *abranch*. All warning messages are suppressed. A *Some comment text* is added to ClearCase as a comment.

6.4.3 CCUnCheckout

6.4.3.1 Description

Task to perform a UnCheckout command to ClearCase.

6.4.3.2 Parameters

Attribute	Values	Required
viewpath	Path to the ClearCase view file or directory that the command will operate on	No
keepcopy	Specifies whether to keep a copy of the file with a .keep extension or not	No

6.4.3.3 Examples

```
<ccuncheckout viewpath="c:/views/viewdir/afile"
        keepcopy="true"/>
```

Does a ClearCase *uncheckout* on the file *c:/views/viewdir/afile*. A copy of the file called *c:/views/viewdir/afile.keep* is kept.

6.4.4 CCUpdate

6.4.4.1 Description

Task to perform an Update command to ClearCase.

6.4.4.2 Parameters

Attribute	Values	Required
viewpath	Path to the ClearCase view file or directory that the command will operate on	No
graphical	Displays a graphical dialog during the update	No
log	Specifies a log file for ClearCase to write to	No
overwrite	Specifies whether to overwrite hijacked files or not	No
rename	Specifies that hijacked files should be renamed with a .keep extension	No
currenttime	Specifies that modification time should be written as the current time. Either currenttime or preservetime can be specified.	No
preservetime	Specifies that modification time should preserved from the VOB time. Either currenttime or preservetime can be specified.	No

6.4.4.3 Examples

```
<ccupdate viewpath="c:/views/viewdir"
        graphical="false"
        log="log.log"
        overwrite="true"
        currenttime="true"
        rename="false"/>
```

Does a ClearCase *update* on the directory *c:/views/viewdir*. A graphical dialog will be displayed. The output will be logged to *log.log* and it will overwrite any hijacked files. The modified time will be set to the current time.

6.5 Depend

A task to manage Java class file dependencies.

6.5.1 Description

The depend task works by determining which classes are out of date with respect to their source and then removing the class files of any other classes which depend on the out-of-date classes.

To determine the class dependencies, the depend task analyses the class files of all class files passed to it. Depend does not parse your source code in any way but relies upon the class references encoded into the class files by the compiler. This is generally faster than parsing the Java source.

To learn more about how this information is obtained from the class files, please refer to the Java Virtual Machine Specification (http://java.sun.com/docs/books/vmspec/)

Since a class' dependencies only change when the class itself changes, the depend task is able to cache dependency information. Only those class files which have changed will have their dependency information re-analysed. Note that if you change a class' dependencies by changing the source, it will be recompiled anyway. You can examine the dependency files created to understand the dependencies of your classes. Please do not rely, however, on the format of the information, as it may change in a later release.

Once depend discovers all of the class dependencies, it "inverts" this relation to determine, for each class, which other classes are dependent upon it. This "affects" list is used to discover which classes are invalidated by the out of date class. The class files of the invalidated classes are removed, triggering the compilation of the affected classes.

The depend task supports an attribute, "closure" which controls whether depend will only consider direct class-class relationships or whether it will also consider transitive, indirect relationships. For example, say there are three classes, A, which depends on B, which in-turn depend on C. Now say that class C is out of date. Without closure, only class B would be removed by depend. With closure set, class A would also be removed. Normally direct relationships are sufficient - it is unusual for a class to depend on another without having a direct relationship. With closure set, you will notice that depend typically removes far more class files.

6.5.2 Performance

The performance of the depend task is dependent on a number of factors such as class relationship complexity and how many class files are out of date. The following tests have been run when build Ant itself, on the author's machine using JDK1.3

Building Ant from clean using Javac	11 seconds
Building Ant from clean using Jikes	5 seconds
Running <depend> when up-to-date and no cache	4 seconds (Depend takes 2 seconds)
Running <depend> when up-to-date and with a cache	2 seconds (Depend takes 1 seconds)

This involves compiling 177 files. The above figures are indicative only. The decision about whether it is cheaper to just recompile all classes or to use the depend task will depend on the size of your project and how interrelated your classes are.

6.5.3 Limitations

There are some dependencies which depend will not detect.

- If the Java compiler optimizes away a class relationship, there can be a source dependency without a class dependency.
- Non public classes cause two problems. Firstly depend cannot relate the class file to a source file. In the future this may be addressed using the source file attribute in the classfile. Secondly, neither depend nor the compiler tasks can detect when a non public class is missing. Inner classes are handled by the depend task.

6.5.4 Parameters

Attribute	Description	Required
srcDir	This is the directory where the source exists. depend will examine this to determine which classes. If you use multiple source directories you can pass this a path of source directories.	Yes
destDir	This is the root directory of the class files which will be analysed. If this is not present, the srcdir is used.	No
cache	This is a directory in which depend can store and retrieve dependency information. If this is not present, depend will not use a cache	No
closure	This attribute controls whether depend only removes classes which directly depend on out of date classes. If this is set to true, depend will traverse the class dependency graph deleting all affected classes. Defaults to false	No

6.5.5 Examples

```
<depend srcdir="${java.dir}"
destdir="${build.classes}"
cache="depcache"
closure="yes"/>
```

In this example classes in the ${build.classes} directory will be removed if they depend on out-of-date classes. Classes are considered out of date with respect to the source in the ${java.dir} directory using the same mechanism as the javac task. In this instance the depend task caches its dependency information in the depcache directory.

6.6 Ant EJB Tasks User Manual

by

- Tim Fennell (tfenne@rcn.com)
- Conor MacNeill (conor@cortexebusiness.com.au)

Version 1.1 - 2000/07/18

6.6.1 Table of Contents

- Introduction [p 125]
- EJB Tasks [p 125]

6.6.2 Introduction

Ant provides a number of optional tasks for developing Enterprise Java Beans (EJBs) (http://java.sun.com/products/ejb). In general these tasks are specific to the particular vendor's EJB Server. At present the tasks support Weblogic (http://www.bea.com) 4.5.1 and 5.1 EJB servers. Over time we expect further optional tasks to support additional EJB Servers.

6.6.3 EJB Tasks

Task	Application Servers
ddcreator [p 125]	Weblogic 4.5.1
ejbc [p 126]	Weblogic 4.5.1
wlrun [p 127]	Weblogic 4.5.1, 5.1 & 6.0
wlstop [p 129]	Weblogic 4.5.1, 5.1 & 6.0
ejbjar [p 130]	Weblogic 5.1 & 6.0

6.6.4 ddcreator

6.6.4.1 Description:

ddcreator will compile a set of Weblogic text-based deployment descriptors into a serialized EJB deployment descriptor. The selection of which of the text-based descriptors are to be compiled is based on the standard Ant include and exclude selection mechanisms.

6.6.4.2 Parameters:

Attribute	Description	Required
descriptors	This is the base directory from which descriptors are selected.	Yes
dest	The directory where the serialised deployment descriptors will be written	Yes
classpath	This is the classpath to use to run the underlying weblogic ddcreator tool. This must include the `weblogic.ejb.utils.DDCreator` class	No

6.6.4.3 Examples

```
<ddcreator descriptors="${dd.dir}"
           dest="${gen.classes}"
           classpath="${descriptorbuild.classpath}">
  <include name="*.txt" />
</ddcreator>
```

6.6.5 ejbc

6.6.5.1 Description:

The ejbc task will run Weblogic's ejbc tool. This tool will take a serialised deployment descriptor, examine the various EJB interfaces and bean classes and then generate the required support classes necessary to deploy the bean in a Weblogic EJB container. This will include the RMI stubs and skeletons as well as the classes which implement the bean's home and remote interfaces.

The ant task which runs this tool is able to compile several beans in a single operation. The beans to be compiled are selected by including their serialised deployment descriptors. The standard ant `include` and `exclude` constructs can be used to select the deployment descriptors to be included.

Each descriptor is examined to determine whether the generated classes are out of date and need to be regenerated. The deployment descriptor is de-serialized to discover the home, remote and implementation classes. The corresponding source files are determined and checked to see their modification times. These times and the modification time of the serialised descriptor itself are compared with the modification time of the generated classes. If the generated classes are not present or are out of date, the ejbc tool is run to generate new versions.

6.6.5.2 Parameters:

Attribute	Description	Required
descriptors	This is the base directory from which the serialised deployment descriptors are selected.	Yes
dest	The base directory where the generated classes, RIM stubs and RMI skeletons are written	Yes
manifest	The name of a manifest file to be written. This manifest will contain an entry for each EJB processed	Yes
src	The base directory of the source tree containing the source files of the home interface, remote interface and bean implementation classes.	Yes
classpath	This classpath must include both the `weblogic.ejbc` class and the classfiles of the bean, home interface, remote interface, etc of the bean being processed.	No

6.6.5.3 Examples

```
<ejbc descriptors="${gen.classes}"
           src="${src.dir}"
           dest="${gen.classes}"
           manifest="${build.manifest}"
           classpath="${descriptorbuild.classpath}">
  <include name="*.ser" />
</ejbc>
```

6.6.6 wlrun

6.6.6.1 Description:

The `wlrun` task is used to start a weblogic server. The task runs a weblogic instance in a separate Java Virtual Machine. A number of parameters are used to control the operation of the weblogic instance. Note that the task, and hence ant, will not complete until the weblogic instance is stopped.

6.6.6.2 Parameters:

Attribute	Description	Required for 4.5.1 and 5.1	Required for 6.0
BEA Home	The location of the BEA Home qwhere the server's config is defined. If this attribute is present, wlrun assumes that the server will be running under Weblogic 6.0	N/A	Yes
home	The location of the weblogic home that is to be used. This is the location where weblogic is installed.	Yes	Yes. Note this is the absolute location, not relative to BEA home.
Domain	The domain to which the server belongs.	N/A	Yes
classpath	The classpath to be used with the Java Virtual Machine that runs the Weblogic Server. Prior to Weblogic 6.0, this is typically set to the Weblogic boot classpath. Under Weblogic 6.0 this should include all the weblogic jars	Yes	Yes
wlclasspath	The weblogic classpath used by the Weblogic Server.	No	N/A
properties	The name of the server's properties file within the weblogic home directory used to control the weblogic instance.	Yes	N/A
name	The name of the weblogic server within the weblogic home which is to be run. This defaults to "myserver"	No	No
policy	The name of the security policy file within the weblogic home directory that is to be used. If not specified, the default policy file `weblogic.policy` is used.	No	No
username	The management username used to manage the server	N/A	No
password	The server's management password	N/A	Yes
pkPassword	The private key password so the server can decrypt the SSL private key file	N/A	No
jvmargs	Additional argument string passed to the Java Virtual Machine used to run the Weblogic instance.	No	No
args	Additional argument string passed to the Weblogic instance.	No	No

6.6.6.3 Nested Elements

The wlrun task supports nested <classpath> and <wlclasspath> elements to set the repsective classpaths.

6.6.6.4 Examples

This example shows the use of wlrun to run a server under Weblogic 5.1

```
<wlrun taskname="myserver"
       classpath="${weblogic.boot.classpath}"
       wlclasspath="${weblogic.classes}:${code.jars}"
       name="myserver"
       home="${weblogic.home}"
       properties="myserver/myserver.properties"/>
```

This example shows wlrun being used to run the petstore server under Weblogic 6.0

```
<wlrun taskname="petstore"
       classpath="${weblogic.classes}"
       name="petstoreServer"
       domain="petstore"
       home="${weblogic.home}"
       password="petstorePassword"
       beahome="${bea.home}"/>
```

6.6.7 wlstop

6.6.7.1 Description:

The `wlstop` task is used to stop a weblogic instance which is currently running. To shut down an instance you must supply both a username and a password. These will be stored in the clear in the build script used to stop the instance. For security reasons, this task is therefore only appropriate in a development environment.

This task works for most version of Weblogic, including 6.0. You need to specify the BEA Home to have this task work correctly under 6.0

6.6.7.2 Parameters:

Attribute	Description	Required
BEAHome	This attribute selects Weblogic 6.0 shutdown.	No
classpath	The classpath to be used with the Java Virtual Machine that runs the Weblogic Shutdown comment.	Yes
user	The username of the account which will be used to shutdown the server	Yes
password	The password for the account specified in the user parameter.	Yes
url	The URL which describes the port to which the server is listening for T3 connections. For example, t3://localhost:7001	Yes
delay	The delay in seconds after which the server will stop. This defaults to an immediate shutdown.	No

6.6.7.3 Nested Element

The classpath of the welstop task can be set by a <classpath> nested element.

6.6.7.4 Examples

This example show the shutdown for a Weblogic 6.0 server

```
<wlstop classpath="${weblogic.classes}"
        user="system"
        url="t3://localhost:7001"
        password="foobar"
        beahome="${bea.home}"/>
```

6.6.8 ejbjar

6.6.8.1 Description:

This task is designed to support building of EJB1.1 jar files. Support is currently provided for 'vanilla' EJB1.1 jar files - i.e. those containing only the user generated class files and the standard deployment descriptor. Nested elements provide support for vendor specific deployment tools. These currently include:

- an element or building Weblogic 5.1/6.0 session/entity beans using the weblogic.ejbc tool.
- an element for building TOPLink for WebLogic 2.5.1-enabled entity beans.

This task supports two approaches to creating ejb jar files. The first approach assumes a particular naming convention for deployment descriptor files. For an Account bean, for example, the deployment descriptor would be named `Account-ejb-jar.xml`. This naming convention allows the task to distinguish deployment descriptors without relying on their positioning within a source tree. It is also used to derive the name of the .jar file which is generated. For the example this would be `Account.jar`. Vendor specific files are assumed to be named in a similar fashion. The deployment descriptor file which defines additional weblogic specific information for the above bean would be `Account-weblogic-ejb-jar.xml`. The second approach does not require a naming convention. This approach uses a specified a jar name for the resultant ejb jar. If the jar name is present, then no naming convention is required. If the jar name is not specified, then the default naming convention is expected for the deployment descriptor files.

The task works as a directory scanning task, and performs an action for each deployment descriptor found. As such the includes and excludes should be set to ensure that all desired EJB1.1 descriptors are found, but no application server descriptors are found. For each descriptor found, ejbjar will parse the deployment descriptor to determine the necessary class files which implement the bean. These files are assembled along with the deployment descriptors into a well formed EJB jar file. Any support files which need to be included in the generated jarcan be added with the <support> nested element. For each class included in the jar, ejbjar will scan for any super classes or super interfaces. These will be added to the generated jar.

If no nested vendor-specific deployment elements are present, the task will simply generate a generic EJB jar. Such jars are typically used as the input to vendor-specific deployment tools. For each nested deployment element, a vendor specific deployment tool is run to generate a jar file ready for deployment in that vendor's EJB container.

The jar files are only built if they are out of date. Each deployment tool element will examine its target jar file and determine if it is out of date with respect to the class files and deployment descriptors that make up the bean. If any of these files are newer than the jar file the jar will be rebuilt otherwise a message is logged that the jar file is up to date.

6.6.8.2 Parameters:

Attribute	Description	Required
descriptordir	The base directory under which to scan for EJB deployment descriptors. If this attribute is not specified, then the deployment descriptors must be located in the directory specified by the 'srcdir' attribute.	No
srcdir	The base directory containing the .class files that make up the bean. Note that this can be the same as the descriptordir if all files are in the same directory tree.	Yes
destdir	The base directory into which generated jar files are deposited. Jar files are deposited in directories corresponding to their location within the descriptordir namespace. Note that this attribute is only used if the task is generating generic jars (i.e. no vendor-specific deployment elements have been specified).	Yes
basejarname	The base name that is used for the generated jar files. If this attribute is specified, the generic jar file name will use this value as the prefix (followed by the value specified in the 'genericjarsuffix' attribute) and the resultant ejb jar file (followed by any suffix specified in the nested element).	No
basenameterminator	String value used to substring out a string from the name of each deployment descriptor found, which is then used to locate related deployment descriptors (e.g. the WebLogic descriptors). For example, a basename of '.' and a deployment descriptor called 'FooBean.ejb-jar.xml' would result in a basename of 'FooBean' which would then be used to find FooBean.weblogic-ejb-jar.xml and FooBean.weblogic-cmp-rdbms-jar.xml, as well as to create the filenames of the jar files as FooBean-generic.jar and FooBean-wl.jar. This attribute is not used if the 'basejarname' attribute is specified.	No, defaults to '-'.
genericjarsuffix	String value appended to the basename of the deployment descriptor to create the filename of the generic EJB jar file.	No, defaults to '-generic.jar'.
classpath	This classpath is used when resolving classes which are to be added to the jar. Typically nested deployment tool elements will also support a classpath which will be combined with this classpath when resolving classes	No.
flatdestdir	Set this attribute to true if you want all generated jars to be placed in the root of the destdir, rather than according to the location of the deployment descriptor within the descriptor dir hierarchy.	No.

6.6.8.3 Nested Elements

In addition to the vendor specific nested elements, the ejbjar task provides three nested elements.

6.6.8.3.1 Classpath

The <classpath> nested element allows the classpath to be set. It is useful when setting the classpath from a reference path. IN all othe rrespects the behaviour is the same as the classpath attribute.

6.6.8.3.2 dtd

The <dtd> element is used to specify the local location of DTDs to be used when parsing the EJB deployment descriptor. Using a local DTD is much faster than loading the DTD across the net. If you are running ejbjar behind a firewall you may not even be able to access the remote DTD. The supported vendor-specific nested elements know the location of the required DTDs within the vendor class hierarchy and, in general, this means <dtd> elements are not required. It does mean, however, that the vendor's class hierarchy must be available in the classpath when Ant is started. If your want to run Ant without requiring the vendor classes in the classpath, you would need to use a <dtd> element.

Attribute	Description	Required
publicId	The public Id of the DTD for which the location is being provided	Yes
location	The location of the local copy of the DTD. This can either be a file or a resource loadable from the classpath.	Yes

6.6.8.3.3 support

The <support> nested element is used to supply additional classes (files) to be included in the generated jars. The <support> element is a FileSet, so it can either reference a fileset declared elsewehere or it can be defined in-place with the approrpiate <include> and <exclude> nested elements. The files in the support fileset are added into the generated EJB jar in the same relative location as their location within the support fileset. Note that when ejbjar generates more than one jar file, the support files are added to each one.

6.6.8.4 Vendor-specific deployment elements

Each vendor-specific nested element controls the generation of a deployable jar specific to that vendor's EJB container. The parameters for each supported deployment element are detailed here.

6.6.8.5 Weblogic element

The weblogic element is used to control the weblogic.ejbc compiler for generating weblogic EJB jars. Prior to Ant 1.3, the method of locating CMP descriptors was to use the ejbjar naming convention. So if your ejb-jar was called, Customer-ejb-jar.xml, your weblogic descriptor was called Customer-weblogic-ejb-jar.xml and your CMP descriptor had to be Customer-weblogic-cmp- rdbms-jar.xml. In addition, the <type-storage> element in the weblogic descriptor had to be set to the standard name META-INF/weblogic-cmp-rdbms- jar.xml, as that is where the CMP descriptor was mapped to in the generated jar.

There are a few problems with this scheme. It does not allow for more than one CMP descriptor to be defined in a jar and it is not compatible with the deployment descriptors generated by some tools.

In Ant 1.3, ejbjar parses the weblogic deployment descriptor to discover the CMP descriptors, which are then included automatically. This behaviour is controlled by the newCMP attribute. Note that if you move to the new method of determining CMP descriptors, you will need to update your weblogic deployment descriptor's <type-storage> element. In the above example, you would define this as META-INF/Customer-weblogic-cmp-rdbms-jar.xml.

Attribute	Description	Required
destdir	The base directory into which the generated weblogic ready jar files are deposited. Jar files are deposited in directories corresponding to their location within the descriptordir namespace.	Yes
genericjarsuffix	A generic jar is generated as an intermediate step in build the weblogic deployment jar. The suffix used to generate the generic jar file is not particularly important unless it is desired to keep the generic jar file. It should not, however, be the same as the suffix setting.	No, defaults to '-generic.jar'.
suffix	String value appended to the basename of the deployment descriptor to create the filename of the WebLogic EJB jar file.	No, defaults to '.jar'.
classpath	The classpath to be used when running the weblogic ejbc tool. Note that this tool typically requires the classes that make up the bean to be available on the classpath. Currently, however, this will cause the ejbc tool to be run in a separate VM	No
wlclasspath	Weblogic 6.0 will give a warning if the home and remote interfaces of a bean are on the system classpath used to run weblogic.ejbc. In that case, the standard weblogic classes should be set with this attribute (or equivalent nested element) and the home and remote interfaces located with the standard classpath attribute	No
keepgeneric	This controls whether the generic file used as input to ejbc is retained.	No, defaults to false
compiler	This allows for the selection of a different compiler to be used for the compilation of the generated Java files. This could be set, for example, to Jikes to compile with the Jikes compiler.	No
rebuild	This flag controls whether weblogic.ejbc is always invoked to build the jar file. In certain circumstances, such as when only a bean class has been changed, the jar can be generated by merely replacing the changed classes and not rerunning ejbc. Setting this to false will reduce the time to run ejbjar.	No, defaults to true.
keepgenerated	Controls whether weblogic will keep the generated Java files used to build the class files added to the jar. This can be useful when debug-ging	No, defaults to false.
args	Any additional arguments to be passed to the weblogic.ejbc tool.	No.

weblogicdtd	**Deprecated**. Defines the location of the ejb-jar DTD in the weblogic class hierarchy. This should not be necessary if you have weblogic in your classpath. If you do not, you should use a nested <dtd> element, described above. If you do choose to use an attribute, you should use the ejbdtd attribute in preference to this one, anyway.	No.
wldtd	**Deprecated**. Defines the location of the weblogic-ejb-jar DTD which covers the Weblogic specific deployment descriptors. This should not be necessary if you have weblogic in your classpath. If you do not, you should use a nested <dtd> element, described above.	No.
ejbdtd	**Deprecated**. Defines the location of the ejb-jar DTD in the weblogic class hierarchy. This should not be necessary if you have weblogic in your classpath. If you do not, you should use a nested <dtd> element, described above.	No.
newCMP	If this is set to true, the ne wmethod for locating CMP descriptors will be used.	No. Defaults to false
oldCMP	**Deprecated** This is an antonym for newCMP which should be used instead.	No.

The weblogic nested element itself supports two nested elements <classpath> and <wlclasspath> which are used to set the respective classpaths. These nested elements are useful when setting up class paths using reference Ids.

6.6.8.6 TOPLink for Weblogic element

The TopLink element is used to handle beans which use Toplink for the CMP operations. It is derived from the standard weblogic element so it supports the same set of attributes please these additional attributes

Attribute	Description	Required
toplinkdescriptor	This specifies the name of the TOPLink deployment descriptor file contained in the 'descriptordir' directory.	Yes
toplinkdtd	This specifies the location of the TOPLink DTD file. This can be a file path or a file URL. This attribute is not required, but using a local DTD is recommended.	No, defaults to dtd file at www.objectpeople.com.

6.6.8.7 Examples

This example shows ejbjar being used to generate deployment jars using a Weblogic EJB container. This example requires the naming standard to be used for the deployment descriptors. Using this format will create a ejb jar file for each variation of '*-ejb-jar.xml' that is found in the deployment descriptor directory.

```
<ejbjar srcdir="${build.classes}"
        descriptordir="${descriptor.dir}">
  <weblogic destdir="${deploymentjars.dir}"
            classpath="${descriptorbuild.classpath}"/>
  <include name="**/*-ejb-jar.xml"/>
  <exclude name="**/*weblogic*.xml"/>
</ejbjar>
```

If weblogic is not in the Ant classpath, the following example shows how to specify the location of the weblogic DTDs. This example also show the use of a nested classpath element.

```
<ejbjar descriptordir="${src.dir}" srcdir="${build.classes}">
   <weblogic destdir="${deployment.webshop.dir}"
             keepgeneric="true"
             args="-g -keepgenerated ${ejbc.compiler}"
             suffix=".jar"
             oldCMP="false">
     <classpath>
       <pathelement path="${descriptorbuild.classpath}"/>
     </classpath>
   </weblogic>
   <include name="**/*-ejb-jar.xml"/>
   <exclude name="**/*-weblogic-ejb-jar.xml"/>
   <dtd publicId="-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 1.1//EN"
        location="${weblogic.home}/classes/weblogic/ejb/deployment/xml/ejb-jar.dtd"/>
   <dtd publicId="-//BEA Systems, Inc.//DTD WebLogic 5.1.0 EJB//EN"
        location="${weblogic.home}/classes/weblogic/ejb/deployment/xml/weblogic-ejb-jar.dtd"/>
</ejbjar>
```

This example shows ejbjar being used to generate a single deployment jar using a Weblogic EJB container. This example does not require the deployment descriptors to use the naming standard. This will create only one ejb jar file - 'TheEJBJar.jar'.

```
<ejbjar srcdir="${build.classes}"
        descriptordir="${descriptor.dir}"
        basejarname="TheEJBJar">
  <weblogic destdir="${deploymentjars.dir}"
            classpath="${descriptorbuild.classpath}"/>
  <include name="**/ejb-jar.xml"/>
  <exclude name="**/weblogic*.xml"/>
</ejbjar>
```

This example shows ejbjar being used to generate deployment jars for a TOPLink-enabled entity bean using a Weblogic EJB container. This example does not require the deployment descriptors to use the naming standard. This will create only one TOPLink-enabled ejb jar file - 'Address.jar'.

```
<ejbjar srcdir="${build.dir}"
        destdir="${solant.ejb.dir}"
        descriptordir="${descriptor.dir}"
        basejarname="Address">
        <weblogictoplink destdir="${solant.ejb.dir}"
                classpath="${java.class.path}"
                keepgeneric="false"
                toplinkdescriptor="Address.xml"
                toplinkdtd="file:///dtdfiles/toplink-cmp_2_5_1.dtd"
```

```
                suffix=".jar"/>
        <include name="**/ejb-jar.xml"/>
        <exclude name="**/weblogic-ejb-jar.xml"/>
</ejbjar>
```

This final example shows how you would set-up ejbjar under Weblogic 6.0. It also shows the use of the <support> element to add support files

```
<ejbjar descriptordir="${dd.dir}" srcdir="${build.classes.server}">
   <include name="**/*-ejb-jar.xml"/>
   <exclude name="**/*-weblogic-ejb-jar.xml"/>
   <support dir="${build.classes.server}">
        <include name="**/*.class"/>
   </support>
   <weblogic destdir="${deployment.dir}"
             keepgeneric="true"
             suffix=".jar"
             rebuild="false">
     <classpath>
        <pathelement path="${build.classes.server}"/>
     </classpath>
     <wlclasspath>
        <pathelement path="${weblogic.classes}"/>
     </wlclasspath>
   </weblogic>
</ejbjar>
```

6.7 FTP

6.7.1 Description

Note: The ftp-task uses the NetComponents-Package which you will need to download from http://www.savarese.org and add to your classpath.

The ftp task implements a basic FTP client that can send, receive, list, and delete files. See below for descriptions and examples of how to perform each task.

The ftp task makes no attempt to determine what file system syntax is required by the remote server, and defaults to Unix standards. *remotedir* must be specified in the exact syntax required by the ftp server. If the usual Unix conventions are not supported by the server, *separator* can be used to set the file separator that should be used instead.

See the section on directory based tasks [p 16] , on how the inclusion/exclusion of files works, and how to write patterns.

6.7.2 Parameters

Attribute	Description	Required
server	the address of the remote ftp server.	Yes
port	the port number of the remote ftp server. Defaults to port 21.	No
userid	the login id to use on the ftp server.	Yes
password	the login password to use on the ftp server.	Yes
remotedir	the directory to which to upload files on the ftp server.	No
action	the ftp action to perform, defaulting to "send". Currently supports "put", "get", "del", and "list".	No
binary	selects binary-mode ("yes") or text-mode ("no") transfers. Defaults to "yes"	No
passive	selects passive-mode ("yes") transfers. Defaults to "no"	No
verbose	displays information on each file transferred if set to "yes". Defaults to "no".	No
depends	transfers only new or changed files if set to "yes". Defaults to "no".	No
newer	a synonym for *depends*.	No
separator	sets the file separator used on the ftp server. Defaults to "/".	No
listing	the file to write results of the "list" action. Required for the "list" action, ignored otherwise.	No

6.7.3 Sending Files

The easiest way to describe how to send files is with a couple of examples:

```
<ftp server="ftp.apache.org"
     userid="anonymous"
     password="me@myorg.com">
  <fileset dir="htdocs/manual"/>
</ftp>
```

Logs in to `ftp.apache.org` as `anonymous` and uploads all files in the `htdocs/manual` directory to the default directory for that user.

```
<ftp server="ftp.apache.org"
     remotedir="incoming"
     userid="anonymous"
     password="me@myorg.com"
     depends="yes"
>
  <fileset dir="htdocs/manual"/>
</ftp>
```

Logs in to `ftp.apache.org` as `anonymous` and uploads all new or changed files in the `htdocs/manual` directory to the `incoming` directory relative to the default directory for `anonymous`.

```
<ftp server="ftp.apache.org"
     port="2121"
     remotedir="/pub/incoming"
     userid="coder"
     password="java1"
     depends="yes"
     binary="no"
>
  <fileset dir="htdocs/manual">
    <include name="**/*.html"/>
  </fileset>
</ftp>
```

Logs in to `ftp.apache.org` at port `2121` as `coder` with password `java1` and uploads all new or changed HTML files in the `htdocs/manual` directory to the `/pub/incoming` directory. The files are transferred in text mode. Passive mode has been switched on to send files from behind a firewall.

```
<ftp server="ftp.nt.org"
     remotedir="c:\uploads"
     userid="coder"
     password="java1"
     separator="\"
     verbose="yes"
```

```
>
  <fileset dir="htdocs/manual">
    <include name="**/*.html"/>
  </fileset>
</ftp>
```

Logs in to the Windows-based `ftp.nt.org` as `coder` with password `java1` and uploads all HTML files in the `htdocs/manual` directory to the `c:\uploads` directory. Progress messages are displayed as each file is uploaded.

6.7.4 Getting Files

Getting files from an FTP server works pretty much the same way as sending them does. The only difference is that the nested filesets use the remotedir attribute as the base directory for the files on the FTP server, and the dir attribute as the local directory to put the files into. The file structure from the FTP site is preserved on the local machine.

```
<ftp action="get"
     server="ftp.apache.org"
     userid="anonymous"
     password="me@myorg.com">
  <fileset dir="htdocs/manual" >
    <include name="**/*.html"/>
  </fileset>
</ftp>
```

Logs in to `ftp.apache.org` as `anonymous` and recursively downloads all .html files from default directory for that user into the `htdocs/manual` directory on the local machine.

6.7.5 Deleting Files

As you've probably guessed by now, you use nested fileset elements to select the files to delete from the remote FTP server. Again, the filesets are relative to the remote directory, not a local directory. In fact, the dir attribute of the fileset is ignored completely.

```
<ftp action="del"
     server="ftp.apache.org"
     userid="anonymous"
     password="me@myorg.com" >
  <fileset>
    <include name="**/*.tmp"/>
  </fileset>
</ftp>
```

Logs in to `ftp.apache.org` as `anonymous` and tries to delete all *.tmp files from the default directory for that user. If you don't have permission to delete a file, a BuildException is thrown.

6.7.6 Listing Files

```
<ftp action="list"
     server="ftp.apache.org"
     userid=quot;anonymous"
     password="me@myorg.com"
     listing="data/ftp.listing" >
  <fileset>
    <include name="**"/>
  </fileset>
</ftp>
```

This provides a file listing in `data/ftp.listing` of all the files on the FTP server relative to the default directory of the `anonymous` user. The listing is in whatever format the FTP server normally lists files.

6.8 JavaCC

6.8.1 Description

Invokes the JavaCC (http://www.metamata.com/javacc/) compiler compiler on a grammar file.

To use the javacc task, set the *target* attribute to the name of the grammar file to process. You also need to specify the directory containing the JavaCC installation using the *javacchome* attribute, so that ant can find the JavaCC classes. Optionally, you can also set the *outputdirectory* to write the generated file to a specific directory. Otherwise javacc writes the generated files to the directory containing the grammar file.

This task only invokes JavaCC if the grammar file is newer than the generated Java files. javacc assumes that the Java class name of the generated parser is the same as the name of the grammar file, ignoring the .jj. If this is not the case, the javacc task will still work, but it will always generate the output files.

6.8.2 Parameters

Attribute	Description	Required
target	The grammar file to process.	Yes
javacchome	The directory containing the JavaCC distribution.	Yes
outputdirectory	The directory to write the generated files to. If not set, the files are written to the directory containing the grammar file.	No
buildparser	Sets the BUILD_PARSER grammar option. This is a boolean option.	No
buildtokenmanager	Sets the BUILD_TOKEN_MANAGER grammar option. This is a boolean option.	No
cachetokens	Sets the CACHE_TOKENS grammar option. This is a boolean option.	No
choiceambiguitycheck	Sets the CHOICE_AMBIGUITY_CHECK grammar option. This is an integer option.	No
commontokenaction	Sets the COMMON_TOKEN_ACTION grammar option. This is a boolean option.	No
debuglookahead	Sets the DEBUG_LOOKAHEAD grammar option. This is a boolean option.	No
debugparser	Sets the DEBUG_PARSER grammar option. This is a boolean option.	No
debugtokenmanager	Sets the DEBUG_TOKEN_MANAGER grammar option. This is a boolean option.	No
errorreporting	Sets the ERROR_REPORTING grammar option. This is a boolean option.	No
forcelacheck	Sets the FORCE_LA_CHECK grammar option. This is a boolean option.	No
ignorecase	Sets the IGNORE_CASE grammar option. This is a boolean option.	No
javaunicodeescape	Sets the JAVA_UNICODE_ESCAPE grammar option. This is a boolean option.	No
lookahead	Sets the LOOKAHEAD grammar option. This is an integer option.	No
optimizetokenmanager	Sets the OPTIMIZE_TOKEN_MANAGER grammar option. This is a boolean option.	No
otherambiguitycheck	Sets the OTHER_AMBIGUITY_CHECK grammar option. This is an integer option.	No
sanitycheck	Sets the SANITY_CHECK grammar option. This is a boolean option.	No
static	Sets the STATIC grammar option. This is a boolean option.	No
unicodeinput	Sets the UNICODE_INPUT grammar option. This is a boolean option.	No
usercharstream	Sets the USER_CHAR_STREAM grammar option. This is a boolean option.	No
usertokenmanager	Sets the USER_TOKEN_MANAGER grammar option. This is a boolean option.	No

6.8.3 Example

```
<javacc
    target="src/Parser.jj"
    outputdirectory="build/src"
    javacchome="c:/program files/JavaCC"
    static="true"
/>
```

This invokes JavaCC on grammar file src/Parser.jj, writing the generated files to build/src. The grammar option STATIC is set to true when invoking JavaCC.

6.9 Javah

6.9.1 Description

Generates JNI headers from a Java class.

When this task executes, it will generate the C header and source files that are needed to implement native methods. JNI operates differently depending on whether JDK1.2 (http://java.sun.com/j2se/1.3/docs/tooldocs/win32/javah.html) (or later) or pre-JDK1.2 (http://java.sun.com/products/jdk/1.1/docs/tooldocs/win32/javah.html) systems are used.

6.9.2 Parameters

Attribute	Description	Required
class	the fully-qualified name of the class (or classes, separated by commas)	Yes
outputFile	concatenates the resulting header or source files for all the classes listed into this file	Yes
destdir	sets the directory where javah saves the header files or the stub files.	
force	specifies that output files should always be written (JDK1.2 only)	No
old	specifies that old JDK1.0-style header files should be generated (otherwise output file contain JNI-style native method function prototypes) (JDK1.2 only)	No
stubs	generate C declarations from the Java object file (used with old)	No
verbose	causes Javah to print a message concerning the status of the generated files	No
classpath	the classpath to use.	No
bootclasspath	location of bootstrap class files.	No
extdirs	location of installed extensions.	No

Either outputFile or destdir must be supplied, but not both.

6.9.3 Examples

```
<javah destdir="c" class="org.foo.bar.Wibble" />
```

makes a JNI header of the named class, using the JDK1.2 JNI model. Assuming the directory 'c' already exists, the file `org_foo_bar_Wibble.h` is created there. If this file already exists, it is left unchanged.

```
<javah outputFile="wibble.h">
  <class name="org.foo.bar.Wibble,org.foo.bar.Bobble"/>
</javah>
```

is similar to the previous example, except the output is written to a file called `wibble.h` in the current directory.

```
<javah destdir="c" force="yes">
  <class name="org.foo.bar.Wibble"/>
  <class name="org.foo.bar.Bobble"/>
  <class name="org.foo.bar.Tribble"/>
</javah>
```

writes three header files, one for each of the classes named. Because the force option is set, these header files are always written when the Javah task is invoked, even if they already exist.

```
<javah destdir="c" verbose="yes" old="yes" force="yes">
  <class name="org.foo.bar.Wibble"/>
  <class name="org.foo.bar.Bobble"/>
  <class name="org.foo.bar.Tribble"/>
</javah>
<javah destdir="c" verbose="yes" stubs="yes" old="yes" force="yes">
  <class name="org.foo.bar.Wibble"/>
  <class name="org.foo.bar.Bobble"/>
  <class name="org.foo.bar.Tribble"/>
</javah>
```

writes the headers for the three classes using the ’old’ JNI format, then writes the corresponding .c stubs. The verbose option will cause Javah to describe its progress.

6.10 JJTree

6.10.1 Description

Invokes the JJTree (http://www.metamata.com/javacc/) preprocessor for the JavaCC compiler compiler. It inserts parse tree building actions at various places in the JavaCC source that it generates. The output of JJTree is run through JavaCC to create the parser.

To use the jjtree task, set the *target* attribute to the name of the jjtree grammar file to process. You also need to specify the directory containing the JavaCC installation using the *javacchome* attribute, so that ant can find the JavaCC classes. Optionally, you can also set the *outputdirectory* to write the generated file to a specific directory. Otherwise jjtree writes the generated JavaCC grammar file to the directory containing the JJTree grammar file.

This task only invokes JJTree if the grammar file is newer than the generated JavaCC file.

6.10.2 Parameters

Attribute	Description	Required
target	The jjtree grammar file to process.	Yes
javacchome	The directory containing the JavaCC distribution.	Yes
outputdirectory	The directory to write the generated file to. If not set, the files are written to the directory containing the grammar file.	No
buildnodefiles	Sets the BUILD_NODE_FILES grammar option. This is a boolean option.	No
multi	Sets the MULTI grammar option. This is a boolean option.	No
nodedefaultvoid	Sets the NODE_DEFAULT_VOID grammar option. This is a boolean option.	No
nodefactory	Sets the NODE_FACTORY grammar option. This is boolean option.	No
nodescopehook	Sets the NODE_SCOPE_HOOK grammar option. This is a boolean option.	No
nodeusesparser	Sets the NODE_USES_PARSER grammar option. This is a boolean option.	No
static	Sets the STATIC grammar option. This is a boolean option.	No
visitor	Sets the VISITOR grammar option. This is a boolean option.	No
nodepackage	Sets the NODE_PACKAGE grammar option. This is a string option.	No
visitorexception	Sets the VISITOR_EXCEPTION grammar option. This is a string option.	No
nodeprefix	Sets the NODE_PREFIX grammar option. This is a string option.	No

6.10.3 Example

```
<jjtree
    target="src/Parser.jjt"
    outputdirectory="build/src"
    javacchome="c:/program files/JavaCC"
    nodeusesparser="true"
/>
```

This invokes JJTree on grammar file src/Parser.jjt, writing the generated grammar file, Parser.jj, file to build/src. The grammar option NODE_USES_PARSER is set to true when invoking JJTree.

6.11 Jlink

6.11.1 Description:

Links entries from sub-builds and libraries.

The jlink task can be used to build jar and zip files, similar to the *jar* task. However, jlink provides options for controlling the way entries from input files are added to the output file. Specifically, capabilities for merging entries from multiple zip or jar files is available.

If a mergefile is specified directly (eg. at the top level of a *mergefiles* pathelement) *and* the mergefile ends in ".zip" or ".jar", entries in the mergefile will be merged into the outfile. A file with any other extension will be added to the output file, even if it is specified in the mergefiles element. Directories specified in either the mergefiles or addfiles element are added to the output file as you would expect: all files in subdirectories are recursively added to the output file with appropriate prefixes in the output file (without merging).

In the case where duplicate entries and/or files are found among the files to be merged or added, jlink merges or adds the first entry and ignores all subsequent entries.

jlink ignores META-INF directories in mergefiles. Users should supply their own manifest information for the output file.

It is possible to refine the set of files that are being jlinked. This can be done with the *includes*, *includesfile*, *excludes*, *excludesfile*, and *defaultexcludes* attributes on the *addfiles* and *mergefiles* nested elements. With the *includes* or *includesfile* attribute you specify the files you want to have included by using patterns. The *exclude* or *excludesfile* attribute is used to specify the files you want to have excluded. This is also done with patterns. And finally with the *defaultexcludes* attribute, you can specify whether you want to use default exclusions or not. See the section on directory based tasks [p 16] , on how the inclusion/exclusion of files works, and how to write patterns. The patterns are relative to the *base* directory.

6.11.2 Parameters:

Attribute	Description	Required
outfile	the path of the output file.	Yes
compress	whether or not the output should be compressed. *true*, *yes*, or *on* result in compressed output. If omitted, output will be uncompressed (inflated).	No
mergefiles	files to be merged into the output, if possible.	At least one of mergefiles or addfiles
addfiles	files to be added to the output.	

6.11.3 Examples

The following will merge the entries in mergefoo.jar and mergebar.jar into out.jar. mac.jar and pc.jar will be added as single entries to out.jar.

```
<jlink compress="false" outfile="out.jar">
   <mergefiles>
     <pathelement path="${build.dir}/mergefoo.jar"/>
     <pathelement path="${build.dir}/mergebar.jar"/>
   </mergefiles>
   <addfiles>
     <pathelement path="${build.dir}/mac.jar"/>
     <pathelement path="${build.dir}/pc.zip"/>
   </addfiles>
</jlink>
```

Suppose the file foo.jar contains two entries: bar.class and barnone/myClass.zip. Suppose the path for file foo.jar is build/tempbuild/foo.jar. The following example will provide the entry tempbuild/foo.jar in the out.jar.

```
<jlink compress="false" outfile="out.jar">
   <mergefiles>
     <pathelement path="build/tempbuild"/>
   </mergefiles>
</jlink>
```

However, the next example would result in two top-level entries in out.jar, namely bar.class and barnone/myClass.zip

```
<jlink compress="false" outfile="out.jar">
   <mergefiles>
     <pathelement path="build/tempbuild/foo.jar"/>
   </mergefiles>
</jlink>
```

6.12 JUnit

6.12.1 Description

This task runs tests from the JUnit testing framework. The latest version of the framework can be found at http://www.junit.org. This task requires JUnit 3.0 or above.

Tests are defined by nested `test` or `batchtest` tags, see nested elements [p 151] .

6.12.2 Parameters

Attribute	Description	Required
printsummary	Print one line statistics for each testcase.	No, default is "off"
fork	Run the tests in a separate VM.	No, default is "off"
haltonerror	Stop the build process if an error occurs during the test run.	No, default is "off"
haltonfailure	Stop the build process if a test fails (errors are considered failures as well).	No, default is "off"
timeout	Cancel the individual tests if the don't finish in the given time (measured in milliseconds). Ignored if fork is disabled.	No
maxmemory	Max amount of memory to allocate to the forked VM (ignored if fork is disabled)	No
jvm	the command used to invoke the Java Virtual Machine, default is 'java'. The command is resolved by java.lang.Runtime.exec(). Ignored if fork is disabled.	No, default "java"
dir	The directory to invoke the VM in. (ignored if fork is disabled)	No

6.12.3 Nested Elements

`junit` supports a nested `<classpath>` element, that represents a PATH like structure [p 12] .

6.12.3.1 jvmarg

If fork is enabled, additional parameters may be passed to the new VM via nested `<jvmarg>` attributes, for example:

```
<junit fork="yes">
  <jvmarg value="-Djava.compiler=NONE"/>
  ...
</junit>
```

would run the test in a VM without JIT.

`<jvmarg>` allows all attributes described in Command line arguments [p 13] .

6.12.3.2 sysproperty

Use nested `<sysproperty>` elements to specify system properties required by the class. These properties will be made available to the VM during the execution of the test (either ANT's VM or the forked VM). The attributes for this element are the same as for environment variables [p 45] .

```
<junit fork="no">
  <sysproperty key="basedir" value="${basedir}"/>
  ...
</junit>
```

would run the test in ANT's VM and make the `basedir` property available to the test.

6.12.3.3 formatter

The results of the tests can be printed in different formats. Output will always be sent to a file unless you set the usefile attribute to false, the name of the file is determined by the name of the test and can be set by the `outfile` attribute of `<test>`.

There are two predefined formatters, one prints the test results in XML format, the other emits plain text. Custom formatters that need to implement `org.apache.tools.ant.taskdefs.optional.junit.JUnitResultFormatter` can be specified.

Attribute	Description	Required
type	Use a predefined formatter (either "xml" or "plain").	Exactly one of these.
classname	Name of a custom formatter class.	
extension	Extension to append to the output filename.	Yes, if classname has been used.
usefile	Boolean that determines whether output should be sent to a file.	No, default true.

6.12.3.4 test

Defines a single test class.

Attribute	Description	Required
name	Name of the test class	Yes
fork	Run the tests in a separate VM. Overrides value set in `<junit>`.	No
haltonerror	Stop the build process if an error occurs during the test run. Overrides value set in `<junit>`.	No
haltonfailure	Stop the build process if a test fails (errors are considered failures as well). Overrides value set in `<junit>`.	No
todir	Directory to write the reports to.	No, default is current directory.
outfile	Base name of the test result. The full filename is determined by this attribute and the extension of `formatter`.	No, default is `TEST-name` using the `name` attribute.
if	Only run test if the named property is set.	No
unless	Only run test if the named property is **not** set.	No

Tests can define their own formatters via nested `<formatter>` elements.

6.12.3.5 batchtest

Define a number of tests based on pattern matching.

`batchtest` collects the included files from any number of nested `<fileset>` [p 20] s. It then generates a test class name for each file that ends in `.java` or `.class`.

Attribute	Description	Required
fork	Run the tests in a separate VM. Overrides value set in `<junit>`.	No
haltonerror	Stop the build process if an error occurs during the test run. Overrides value set in `<junit>`.	No
haltonfailure	Stop the build process if a test fails (errors are considered failures as well). Overrides value set in `<junit>`.	No
todir	Directory to write the reports to.	No, default is current directory.
if	Only run tests if the named property is set.	No
unless	Only run tests if the named property is **not** set.	No

Batchtests can define their own formatters via nested `<formatter>` elements.

6.12.4 Examples

```
<junit>
  <test name="my.test.TestCase" />
</junit>
```

Runs the test defined in `my.test.TestCase` in the same VM. No output will be generated unless the test fails.

```
<junit printsummary="yes" fork="yes" haltonfailure="yes">
  <formatter type="plain" />
  <test name="my.test.TestCase" />
</junit>
```

Runs the test defined in `my.test.TestCase` in a separate VM. At the end of the test a single line summary will be printed. A detailed report of the test can be found in `TEST-my.test.TestCase.txt`. The build process will be stopped if the test fails.

```
<junit printsummary="yes" haltonfailure="yes">
  <classpath>
    <pathelement location="${build.tests}" />
    <pathelement path="${java.class.path}" />
  </classpath>

  <formatter type="plain" />

  <test name="my.test.TestCase" haltonfailure="no" outfile="result" >
    <formatter type="xml" />
  </test>

  <batchtest fork="yes" todir="${reports.tests}">
    <fileset dir="${src.tests}">
      <include name="**/*Test*.java" />
      <exclude name="**/AllTests.java" />
    </fileset>
  </batchtest>
</junit>
```

Runs `my.test.TestCase` in the same VM (ignoring the given CLASSPATH), only a warning is printed if this test fails. In addition to the plain text test results, for this test a XML result will be output to `result.xml`.

For each matching file in the directory `${src.tests}` a test is run in a separate VM. If a test fails, the build process is aborted. Results are collected in files named `TEST-`*`name`*`.txt` and written to `${reports.tests}`.

6.13 JUnitReport

Merge the individual XML files generated by the JUnit task and eventually apply a stylesheet on the resulting merged document to provide a browsable report of the testcases results.

6.13.1 Requirements

You need Xalan 1 (http://xml.apache.org/xalan/) to use this task or use the Xalan1 compatibility jar of Xalan2 (http://xml.apache.org/xalan-j/).

6.13.2 Parameters

Attribute	Description	Required
tofile	The name of the XML file that will aggregate all individual XML testsuite previously generated by the JUnit task.	No. Default to TESTS-TestSuites.xml
todir	The directory where should be written the file resulting from the individual XML testsuite aggregation.	No. Default to current directory

6.13.3 Nested Elements

6.13.3.1 fileset

`junitreport` collects individual xml files generated by the JUnit task using the nested `<FileSet>` [p 20] element.

6.13.3.2 report

Generate a browsable report based on the document created by the merge.

6.13.4 Parameters

Attribute	Description	Required
format	The format of the generated report. Must be "noframes" or "frames".	No, default to "frames"
styledir	The directory where the stylesheets are defined. They must be conforming to the following conventions: For the noframes format, only the stylesheet overview-summary.xsl is used. For the frames format, the result will be javadoc-like. List of package will be in top left frame, the list of classes in bottom left frame and the details/summary in the main window. It uses the following XSLs: • `all-packages.xsl` - creates an *all-packages.html* file that contains the list of all packages (top-left frame) with links to their corresponding *package-summary.html* files in their respective location (package name matches path name) • `all-classes.xsl` - creates an *all-classes.html* file that contains the whole list of classes in all packages with links to their corresponding *<classname>-details.html* in their respective package directory. • `overview-packages.xsl` - creates an *overview-packages.html* that makes a summary of all packages and classes not belonging to a package. • `class-details.xsl` - creates a detailed page about the testcases of a testsuite. This this xsl will be applied on all testsuites and written to the *<classname>-details.html* file located in their respective package directory. • `package-summary.xsl` - create a testsuite summary for a package. This will be written to *package-summary.html* in its corresponding package directory. • `classes-list.xsl` - creates the list of classes belonging to a specific package. This will be written to classes-list.html in its corresponding package directory.	No. Default to embedded stylesheets.
todir	The directory where the files resulting from the transformation should be written to.	No. Default to current directory
extension	The extension of the generated files, they must include the dot (ex: .html)	No. Default to .html

6.13.5 Example of report

```
<junitreport todir="./reports">
  <fileset dir="./reports">
    <include name="TEST-*.xml"/>
  </fileset>
  <report format="frames" todir="./report/html"/>
</junitreport>
```

would generate a `TESTS-TestSuites.xml` file in the directory `reports` and generate the default framed report in the directory `report/html`.

6.14 MParse

by

- Stephane Bailliez (sbailliez@imediation.com)

6.14.1 Requirements

This task requires Metamata Development environment 2.0 freely available at Metamata (http://www.metamata.com).

6.14.2 Description

Invokes the Metamata MParse (http://www.metamata.com/parse.html) compiler compiler on a grammar file.

To use the *mparse* task, set the *target* attribute to the name of the grammar file to process. You also need to specify the directory containing the Metamata installation using the *metamatahome* attribute, so that Ant can find the MParse classes.

This task only invokes MParse if the grammar file is newer than the generated Java files. MParse assumes that the Java class name of the generated parser is the same as the name of the grammar file, less the .jj extension.

For additional information about MParse, please consult the online manual available here (http://download.metamata.com/parse.pdf) (PDF)

6.14.3 Parameters

Attribute	Description	Required
target	The .jj grammar file to process. It will only be processed if the grammar is newer than the corresponding .java file.	Yes
metamatahome	The home directory containing the Metamata distribution.	Yes
verbose	Enable all messages	No
debugparser	Enables parser debugging	No
debugscanner	Enables scanner debugging	No
cleanup	Remove the intermediate Sun JavaCC file created during the transformation of the grammar file.	No. Default to false
maxmemory	Set the maximum memory for the JVM. this is a convenient way to set the -mx or -Xmx argument.	No

6.14.4 Nested elements

6.14.4.1 jvmarg

Additional parameters may be passed to the VM via nested `<jvmarg>` attributes. `<jvmarg>` allows all attributes described in Command line arguments [p 13] .

6.14.4.2 classpath

Sets class path (also source path unless one explicitly set). Overrides METAPATH/CLASSPATH environment variables. The `classpath` element represents a PATH like structure [p 12] .

6.14.4.3 sourcepath

Sets source path. Overrides the SOURCEPATH environment variable. The `sourcepath` element represents a PATH like structure [p 12] .

6.14.5 Example

```
<mparse target="c:/metamata/examples/parseexamples/javagrammars/singlefile/JavaParser.jj"
        metamatahome="c:/metamata" cleanup="true">
  <classpath>
    <pathelement location="c:/metamata/examples/"/>
  </classpath>
</mparse>
```

This invokes Metamata MParse installed in `c:/metamata` on one of the grammar file example (`JavaParser.jj`) and cleans up the intermediate Sun JavaCC file.

6.15 Native2Ascii

6.15.1 Description:

Converts files from native encodings to ASCII with escaped Unicode. A common usage is to convert source files maintained in a native operating system encoding, to ASCII prior to compilation.

Files in the directory *src* are converted from a native encoding to ASCII. By default, all files in the directory are converted. However, conversion may be limited to selected files using *includes* and *excludes* attributes. For more information on file matching patterns, see the section on directory based tasks [p 16] . If no *encoding* is specified, the default encoding for the JVM is used. If *ext* is specified, then output files are renamed to use it as a new extension. More sophisticated file name translations can be achieved using a nested <*mapper*> element. By default an identity mapper [p 21] will be used. If *dest* and *src* point to the same directory, the *ext* attribute or a nested <*mapper*> is required.

This task forms an implicit File Set [p 20] , and supports all attributes of `<fileset>` (`dir` becomes `src`) as well as nested `<include>`, `<exclude>`, and `<patternset>` elements.

Attribute	Description	Required
reverse	Reverse the sense of the conversion, i.e. convert from ASCII to native	No
encoding	The native encoding the files are in (default is the default encoding for the JVM)	No
src	The directory to find files in (default is *basedir*)	No
dest	The directory to output file to	Yes
ext	File extension to use in renaming output files	No
defaultexcludes	indicates whether default excludes should be used or not ("yes"/"no"). Default excludes are used when omitted.	No
includes	comma separated list of patterns of files that must be included. All files are included when omitted.	No
includesfile	the name of a file. Each line of this file is taken to be an include pattern	No
excludes	comma separated list of patterns of files that must be excluded. No files (except default excludes) are excluded when omitted.	No
excludesfile	the name of a file. Each line of this file is taken to be an exclude pattern	No

6.15.2 Examples

```
<native2ascii encoding="EUCJIS" src="srcdir" dest="srcdir"
   includes="**/*.eucjis" ext=".java" />
```

Converts all files in the directory *srcdir* ending in `.eucjis` from the EUCJIS encoding to ASCII and renames them to end in `.java`.

```
<native2ascii encoding='EUCJIS" src="native/japanese" dest="src"
   includes="**/*.java />
```

Converts all the files ending in `.java` in the directory *native/japanese* to ASCII, placing the results in the directory *src*. The names of the files remain the same.

6.16 NetRexxC

6.16.1 Description

Compiles a NetRexx (http://www2.hursley.ibm.com/netrexx) source tree within the running (Ant) VM.

The source and destination directory will be recursively scanned for NetRexx source files to compile. Only NetRexx files that have no corresponding class file or where the class file is older than the java file will be compiled.

Files in the source tree are copied to the destination directory, allowing support files to be located properly in the classpath. The source files are copied because the NetRexx compiler cannot produce class files in a specific directory via parameters

The directory structure of the source tree should follow the package hierarchy.

It is possible to refine the set of files that are being compiled/copied. This can be done with the *includes*, *includesfile*, *excludes*, *excludesfile* and *defaultexcludes* attributes. With the *includes* or *includesfile* attribute you specify the files you want to have included by using patterns. The *exclude* or *excludesfile* attribute is used to specify the files you want to have excluded. This is also done with patterns. And finally with the *defaultexcludes* attribute, you can specify whether you want to use default exclusions or not. See the section on directory based tasks [p 16] , on how the inclusion/exclusion of files works, and how to write patterns.

This task forms an implicit FileSet [p 20] and supports all attributes of `<fileset>` (`dir` becomes `srcdir`) as well as the nested `<include>`, `<exclude>` and `<patternset>` elements.

6.16.2 Parameters

Attribute	Description	Required
binary	Whether literals are treated as the java binary type rather than the NetRexx types	No
classpath	The classpath to use during compilation	No
comments	Whether comments are passed through to the generated java source	No
compact	Whether error messages come out in compact or verbose format	No
compile	Whether the NetRexx compiler should compile the generated java code	No
console	Whether or not messages should be displayed on the ’console’	No
crossref	Whether variable cross references are generated	No
decimal	Whether decimal arithmetic should be used for the NetRexx code	No
defaultexcludes	indicates whether default excludes should be used or not ("yes"/"no"). Default excludes are used when omitted.	No

destDir	the destination directory into which the NetRexx source files should be copied and then compiled	Yes
diag	Whether diagnostic information about the compile is generated	No
excludes	comma separated list of patterns of files that must be excluded. No files (except default excludes) are excluded when omitted.	No
excludesfile	the name of a file. Each line of this file is taken to be an exclude pattern	No
explicit	Whether variables must be declared explicitly before use	No
format	Whether the generated java code is formatted nicely or left to match NetRexx line numbers for call stack debugging	No
includes	comma separated list of patterns of files that must be included. All files are included when omitted.	No
includesfile	the name of a file. Each line of this file is taken to be an include pattern	No
java	Whether the generated java code is produced	No
keep	Sets whether the generated java source file should be kept after compilation. The generated files will have an extension of .java.keep, **not** .java	No
logo	Whether the compiler text logo is displayed when compiling	No
replace	Whether the generated .java file should be replaced when compiling	No
savelog	Whether the compiler messages will be written to NetRexxC.log as well as to the console	No
sourcedir	Tells the NetRexx compiler to store the class files in the same directory as the source files. The alternative is the working directory	No
srcDir	Set the source dir to find the source NetRexx files	Yes
strictargs	Tells the NetRexx compiler that method calls always need parentheses, even if no arguments are needed, e.g. `aStringVar.getBytes` vs. `aStringVar.getBytes()`	No
strictassign	Tells the NetRexx compile that assignments must match exactly on type	No
strictcase	Specifies whether the NetRexx compiler should be case sensitive or not	No
strictimport	Whether classes need to be imported explicitly using an `import` statement. By default the NetRexx compiler will import certain packages automatically	No
strictprops	Whether local properties need to be qualified explicitly using `this`	No
strictsignal	Whether the compiler should force catching of exceptions by explicitly named types	No
symbols	Whether debug symbols should be generated into the class file	No
time	Asks the NetRexx compiler to print compilation times to the console	No
trace	Turns on or off tracing and directs the resultant trace output	No
utf8	Tells the NetRexx compiler that the source is in UTF8	No
verbose	Whether lots of warnings and error messages should be generated	Yes

6.16.3 Examples

```
<netrexxc srcDir="/source/project" includes="vnr/util/*"
destDir="/source/project/build" class-
path="/source/project2/proj.jar"  comments="true"  crossref="false"
replace="true" keep="true"/>
```

6.17 Perforce Tasks User Manual

by

- Les Hughes (leslie.hughes@rubus.com)

Version 1.1 - 2001/01/09

6.17.1 Contents

- Introduction [p 164]
- The Tasks [p 164]
- Change History [p 168]

6.17.2 Introduction

These tasks provide an interface to the Perforce (http://www.perforce.com) SCM. The `org.apache.tools.ant.taskdefs.optional.perforce` package consists of a simple framework to support p4 functionality as well as some Ant tasks encapsulating frequently used (by me :-) p4 commands. However, the addition of new p4 commands is a pretty simple task (see the source). Although it is possible to use these commands on the desktop, they were primarily intended to be used by automated build systems.

Note: These tasks require the oro 2.0 (http://jakarta.apache.org/oro) regular expression package. Simply download this package and copy the jakarta-oro-2.0.jar file into Ant's lib directory. You will also need the perforce client executable (p4 or p4.exe but not p4win.exe) in your path.

6.17.3 The Tasks

P4Sync [p 166]	Synchronise a workspce to a depot
P4Change [p 166]	Request a new changelist from the perforce server
P4Edit [p 167]	Open files for edit (checkout)
P4Submit [p 167]	Submit a changelist to the perforce server (checkin)
P4Have [p 167]	List current files in client view, useful for reporting
P4Label [p 168]	Create a label reflecting files in the current workspace

6.17.3.1 General P4 Properties

Each p4 task requires a number of settings, either through build-wide properties, individual attributes or environment variables. These are

Property	Attribute	Env Var	Description	Default
p4.port	port	P4PORT	The p4d server and port to connect to	perforce:1666
p4.client	client	P4CLIENT	The p4 client spec to use	The logged in username
p4.user	user	P4USER	The p4 username	The logged in username
--	view	--	The client, branch or label view to operate upon. See the p4 user guide for more info.	//...

Your local installation of perforce may require other settings (e.g. P4PASSWD, P4CONFIG). At the moment, these can only be set outside of Ant, as environment variables.

6.17.3.2 Examples

Setting in the environment:-

(Unix csh)

```
setenv P4PORT myperforcebox:1666
```

(Unix sh et al)

```
P4USER=myp4userid; export P4USER
```

Using build properties:-

```
<property name="p4.client" value="nightlybuild"/>
```

Using task attributes:-

```
<p4Whatever
        port="myserver:1666"
        client="smoketest"
        user="smoketestdude"
        .
        .
        .
/>
```

For more information regarding the underlying ’p4’ commands you are referred to the Perforce Command Reference available from the Perforce website (http://www.perforce.com/).

6.17.3.3 Taskdefs

Standard taskdefs (for you to copy’n’paste)

```
<taskdef name="p4sync" classname="org.apache.tools.ant.taskdefs.optional.perforce.P4Sync" />
<taskdef name="p4change" classname="org.apache.tools.ant.taskdefs.optional.perforce.P4Change" />
<taskdef name="p4edit" classname="org.apache.tools.ant.taskdefs.optional.perforce.P4Edit" />
<taskdef name="p4submit" classname="org.apache.tools.ant.taskdefs.optional.perforce.P4Submit" />
<taskdef name="p4have" classname="org.apache.tools.ant.taskdefs.optional.perforce.P4Have" />
<taskdef name="p4label" classname="org.apache.tools.ant.taskdefs.optional.perforce.P4Label" />
```

6.17.4 Task Descriptions

6.17.5 P4Sync

6.17.5.1 Description:

Synchronize the current workspace with the depot.

6.17.5.2 Parameters

Attribute	Description	Required
force	force a refresh of files	no
label	sync client to label	no

6.17.5.3 Examples

```
<p4sync label="nightlybuild-0.0123" force="yes" />
<p4sync view="//depot/projects/projectfoo/main/src/..." />
```

6.17.6 P4Change

6.17.6.1 Description:

Request a new changelist from the perforce server. This task sets the ${p4.change} property which can then be passed to P4Submit or P4Edit.

6.17.6.2 Parameters

Attribute	Description	Required
None	--	--

6.17.6.3 Examples

```
<p4change />
```

6.17.7 P4Edit

6.17.7.1 Description:

Open file(s) for edit. P4Change should be used to obtain a new changelist for P4Edit as, although P4Edit can open files to the default change, P4Submit cannot yet submit it.

6.17.7.2 Parameters

Attribute	Description	Required
view	The filespec to request to edit	Yes
change	An existing changelist number to assign files to.	No, but see above.

6.17.7.3 Examples

```
<p4edit
        view="//depot/projects/projectfoo/main/src/Blah.java..."
        change="${p4.change}" />
```

6.17.8 P4Submit

6.17.8.1 Description:

Submit a changelist, usually obtained from P4Change.

6.17.8.2 Parameters

Attribute	Description	Required
change	The changelist number to submit	Yes

6.17.8.3 Examples

```
<p4submit change="${p4.change}" />
```

6.17.9 P4Have

6.17.9.1 Description:

List handy file info reflecting the current client contents.

6.17.9.2 Parameters

Attribute	Description	Required
None	--	--

6.17.9.3 Examples

```
<p4have />
```

6.17.10 P4Label

6.17.10.1 Description:

Create a new label and set contents to reflect current client file revisions.

6.17.10.2 Parameters

Attribute	Description	Required
name	The name of the label	Yes
view	client view to use for label	No
desc	Label Description	No

6.17.10.3 Examples

```
<p4label
        name="NightlyBuild:${DSTAMP}:${TSTAMP}"
        desc="Auto Nightly Build"
/>
```

6.17.11 Change History

Sept 2000	--	Internal Release within Rubus
Nov 2000	V1.0	Initial Release donated to ASF :-)
Jan 2001	V1.1	fixed cross platform (NT/Unix) bug refactored p4 output handling code refactored exec’ing code

6.18 Ant PropertyFile Task User Manual

by

- Thomas Christen (chr@active.ch)
- Jeremy Mawson (jem@loftinspace.com/au (mailto:jem@loftinspace.com.au))

Version 1.1 - 2001/01/28

6.18.1 Table of Contents

- Introduction [p 169]
- PropertyFile Task [p 169]
- Entry Task [p 169]

6.18.2 Introduction

Ant provides an optional task for editing property files. This is very useful when wanting to make unattended modifications to configuration files for application servers and applications. Currently, the task maintains a working property file with the ability to add properties or make changes to existing ones. However, any comments are lost. Work is being done to make this task a bit more "human friendly".

6.18.3 PropertyFile Task

6.18.3.1 Parameters

Attribute	Description	Required
file	Location of the property file to be edited	Yes
comment	Header for the file itself	no

6.18.3.2 Parameters specified as nested elements

6.18.3.2.1 Entry

Use nested `<entry>` elements to specify actual modifcations to the property file itself

Attribute	Description	Required
key	Name of the property name/value pair	Yes
value	Value to set (=), to add (+) or subtract (-)	Yes
type	Regard the value as : int, date or string (default)	No
operation	"+" or "=" (default) for all datatypes "-" (for date and int only).	No
default	Initial value to set for a property if it is not already defined in the property file. For type date, two additional keywords are allowed: "now" or "never".	No
pattern	For int and date type only. If present, Values will be parsed and formatted accordingly.	No

6.18.3.3 Examples

The following changes the my.properties file. Assume my.properties look like:

```
# A comment
akey=novalue
```

After running, the file would now look like

```
#Thu Nov 02 23:41:47 EST 2000
akey=avalue
adate=2000/11/02 23\:41
anint=1
formated.int=0014
formated.date=028 17\:34
```

The slashes conform to the expectations of the Properties class. The file will be stored in a manner so that each character is examined and escaped if necessary. Note that the original comment is now lost. Please keep this in mind when running this task against heavily commented properties files. It may be best to have a commented version in the source tree, copy it to a deployment area, and then run the modifications on the copy. Future versions of PropertyFile will hopefully eliminate this shortcoming.

```
  <propertyfile
      file="my.properties"
      comment"My properties" >
    <entry  key="akey" value="avalue" />
    <entry  key="adate" type="date" value="now"/>
    <entry  key="anint" type="int" operation="+"/>
    <entry  key="formated.int" type="int" default="0013" operation="+" pattern="0000"/>
    <entry  key="formated.date" type="date" value="now" pattern="DDD HH:mm"/>
  </propertyfile>
```

To produce dates relative from today :

```
<propertyfile
    file="my.properties"
    comment="My properties" >
  <entry  key="formated.date-1"
      type="date" default="now" pattern="DDD"
      operation="-" value="1"/>
  <entry  key="formated.tomorrow"
      type="date" default="now" pattern="DDD"
      operation="+" value="1"/>
</propertyfile>
```

Concatenation of strings :

```
<propertyfile
    file="my.properties"
    comment="My properties" >
  <entry  key="progress" default="" operation="+" value="."/>
</propertyfile>
```

Each time called, a "." will be appended to "progress"

6.19 RenameExtensions

6.19.1 Deprecated

This task has been deprecated. Use the move [p 74] task with a glob mapper [p 22] instead.

6.19.2 Description

Renames files in the `srcDir` directory ending with the `fromExtension` string so that they end with the `toExtension` string. Files are only replaced if `replace` is true

See the section on directory based tasks [p 16] , on how the inclusion/exclusion of files works, and how to write patterns. This task forms an implicit FileSet [p 20] and supports all attributes of `<fileset>` (`dir` becomes `srcDir`) as well as the nested `<include>`, `<exclude>` and `<patternset>` elements.

6.19.3 Parameters

Attribute	Description	Required
defaultexcludes	indicates whether default excludes should be used or not ("yes"/"no"). Default excludes are used when omitted.	No
excludes	comma separated list of patterns of files that must be excluded. No files (except default excludes) are excluded when omitted.	No
excludesfile	the name of a file. Each line of this file is taken to be an exclude pattern	No
fromExtention	The string that files must end in to be renamed	Yes
includes	comma separated list of patterns of files that must be included. All files are included when omitted.	No
includesfile	the name of a file. Each line of this file is taken to be an include pattern	No
replace	Whether the file being renamed to should be replaced if it already exists	No
srcDir	The starting directory for files to search in	Yes
toExtension	The string that renamed files will end with on completion	Yes

6.19.4 Examples

```
<renameext srcDir="/source/project1" includes="**"
excludes="**/samples/*" fromExtension=".java.keep" toExten-
sion=".java" replace="true"/>
```

6.20 Script

6.20.1 Description

Execute a script in a BSF (http://oss.software.ibm.com/developerworks/opensource/bsf/) supported language.

All items (tasks, targets, etc) of the running project are accessible from the script, using either their `name` or `id` attributes.

Scripts can do almost anything a task written in Java could do.

6.20.2 Parameters

Attribute	Description	Required
language	The programming language the script is written in. Must be a supported BSF language	No
src	The location of the script as a file, if not inline	No

6.20.3 Examples

```
<project name="squares" default="main" basedir=".">

  <target name="setup">

    <script language="javascript"> <![CDATA[

      for (i=1; i<=10; i++) {
        echo = squares.createTask("echo");
        main.addTask(echo);
        echo.setMessage(i*i);
      }

    ]]> </script>

  </target>

  <target name="main" depends="setup"/>

</project>
```

generates

```
setup:

main:
1
4
9
```

```
  16
  25
  36
  49
  64
  81
  100

  BUILD SUCCESSFUL
```

Another example, using references by id [p 14] and two different scripting languages:

```
  <project name="testscript" default="main">
    <target name="sub">
      <echo id="theEcho"/>
    </target>

    <target name="sub1">
      <script language="netrexx"><![CDATA[
        theEcho.setMessage("In sub1")
        sub.execute
      ]]></script>
    </target>

    <target name="sub2">
      <script language="javascript"><![CDATA[
        theEcho.setMessage("In sub2");
        sub.execute();
      ]]></script>
    </target>

    <target name="main" depends="sub1,sub2"/>
  </project>
```

generates

```
  sub1:
  In sub1

  sub2:
  In sub2

  main:

  BUILD SUCCESSFUL
```

6.21 Sound

6.21.1 Description

Plays a sound-file at the end of the build, according to whether the build failed or succeeded. You can specify either a specific sound-file to play, or, if a directory is specified, the `<sound>` task will randomly select a file to play. Note: At this point, the random selection is based on all the files in the directory, not just those ending in appropriate suffixes for sound-files, so be sure you only have sound-files in the directory you specify.

6.21.2 Parameters

Attribute	Description	Required
loops	the number of extra times to play the sound-file; default is `0`.	No
duration	the amount of time (in milliseconds) to play the sound-file.	No

To specify the sound-files or the sound-file directories, use the nested `<success>` and `<fail>` elements:

```
<success>     the path to a sound-file directory, or the name of a
              specific sound-file, to be played if the build succeeded.
<fail>        the path to a sound-file directory, or the name of a
              specific sound-file, to be played if the build succeeded.
```

6.21.3 Examples

```
<target name="fun" if="fun" unless="fun.done">
  <sound loops="2">
    <success source="${user.home}/sounds/bell.wav"/>
    <fail source="${user.home}/sounds/ohno.wav"/>
  </sound>
  <property name="fun.done" value="true"/>
</target>
```

plays the `bell.wav` sound-file if the build succeeded, or the `ohno.wav` sound-file if the build failed, three times, if the `fun` property is set to `true`. If the target is a dependency of an "initialization" target that other targets depend on, the `fun.done` property prevents the target from being executed more than once.

```
<target name="fun" if="fun" unless="fun.done">
  <sound>
    <success source="//intranet/sounds/success"/>
    <fail source="//intranet/sounds/failure"/>
  </sound>
  <property name="fun.done" value="true"/>
</target>
```

randomly selects a sound-file to play when the build succeeds or fails.

6.22 Stylebook

6.22.1 Description

This executes the apache Stylebook documentation generator. Unlike the commandline version of this tool, all three arguments are required to run stylebook.

6.22.2 Parameters

Attribute	Description	Required
book	the book xml file that the documentation generation starts from	Yes
skindirectory	the directory that contains the stylebook skin	Yes
targetdirectory	the destination directory where the documentation is generated	Yes

The user can also specify the nested <classpath> element which defines classpath in which the task is executed.

6.22.3 Examples

```
<stylebook targetdirectory="build/docs"
           book="src/xdocs/book.xml"
           skindirectory="src/skins/myskin"/>
```

The above will generate documentation in build/docs starting from the book src/xdocs/book.xml and using the skin located in directory src/skins/myskin.

6.23 Telnet

6.23.1 Description

Task to automate a remote telnet session. The task uses nested <read> to indicate strings to wait for, and <write> tags to specify text to send.

If you do specify a userid and password, the system will assume a common unix prompt to wait on. This behavior can be easily over-ridden.

6.23.2 Parameters

Attribute	Values	Required
userid	the login id to use on the telnet server.	No
password	the login password to use on the telnet server.	No
server	the address of the remote telnet server.	Yes
port	the port number of the remote telnet server. Defaults to port 23.	No
initialCR	send a cr after connecting ("yes"). Defaults to "no".	No
timeout	set a default timeout to wait for a response. Specified in seconds. Default is no timeout.	No

6.23.3 Examples

A simple example of connecting to a server and running a command. This assumes a prompt of "ogin:" for the userid, and a prompt of "assword:" for the password.

```
<telnet userid="bob" password="badpass" server="localhost">
   <read>/home/bob</read>
   <write>ls</write>
   <read string="/home/bob"/>
</telnet>
```

This task can be rewritten as:

```
<telnet server="localhost">
   <read>ogin:</read>
   <write>bob</write>
   <read>assword:</read>
   <write>badpass</write>
   <read>/home/bob</read>
   <write>ls</write>
   <read>/home/bob</read>
</telnet>
```

A timeout can be specified at the <telnet> level or at the <read> level. This will connect, issue a sleep command that is suppressed from displaying and wait 10 seconds before quitting.

```
<telnet userid="bob" password="badpass" server="localhost" timeout="20">
   <read>/home/bob</read>
   <write echo="false">sleep 15</write>
   <read timeout="10">/home/bob</read>
</telnet>
```

The task can be used with other ports as well:

```
<telnet port="80" server="localhost" timeout="20">
   <read/>
   <write>GET / http/0.9</write>
   <write/>
   <read timeout="10">&lt;/HTML&gt;</read>
</telnet>
```

6.24 Test

6.24.1 Description

This is a primitive task to execute a unit test in the org.apache.testlet framework.

6.24.2 Parameters

Attribute	Description	Required
showSuccess	a boolean value indicating whether tests should display a message on success	No
showBanner	a boolean value indicating whether a banner should be displayed when starting testlet engine	No
forceShowTrace	a boolean indicating that a stack trace is displayed on any failure	No
showTrace	a boolean indicating that a stack trace is displayed on error (but not normal failure)	No

The user can also specify the nested <classpath> element which defines classpath in which the task is executed. The user also specifies a subelement per testlet executed which has content that specifies tasklet classname.

6.24.3 Examples

```
<test showSuccess="false" showBanner="false" showTrace="true" forceShowTrace="true">
   <classpath refid="test.classpath"/>
   <testlet>org.foo.MyTestlet<testlet/>
   <testlet>org.foo.MyOtherTestlet<testlet/>
<test/>
```

The above will run the testlets org.foo.MyTestlet and org.foo.MyOtherTestlet

6.25 VssGet

6.25.1 Description

Task to perform GET commands to Microsoft Visual Source Safe.

If you specify two or more attributes from version, date and label only one will be used in the order version, date, label.

6.25.2 Parameters

Attribute	Values	Required
login	username,password	No
vsspath	SourceSafe path	Yes
localpath	Override the working directory and get to the specified path	No
writable	true or false	No
recursive	true or false	No
version	a version number to get	No
date	a date stamp to get at	No
label	a label to get for	No
ssdir	directory where `ss.exe` resides. By default the task expects it to be in the PATH.	No

Note that only one of version, date or label should be specified

6.25.3 Examples

```
<vssget localPath="C:\mysrc\myproject"
        recursive="true"
        label="Release1"
        login="me,mypassword"
        vsspath="/source/aProject"
        writable="true"/>
```

Does a get on the VSS-Project *$/source/aproject* using the username *me* and the password *mypassword*. It will recursively get the files which are labeled *Release1* and write them to the local directory *C:\mysrc\myproject*. The local files will be writable.

6.26 VssLabel

6.26.1 Description

Task to perform LABEL commands to Microsoft Visual Source Safe.

Assigns a label to the specified version or current version of a file or project.

6.26.2 Parameters

Attribute	Values	Required
login	username,password	No
vsspath	SourceSafe path	Yes
ssdir	directory where `ss.exe` resides. By default the task expects it to be in the PATH.	No
label	A label to apply to the hierarchy	Yes
version	An existing file or project version to label. By default the current version is labelled.	No

6.26.3 Examples

```
<vsslabel vsspath="/source/aProject"
          login="me,mypassword"
          label="Release1"/>
```

Labels the current version of the VSS project *$/source/aproject* with the label *Release1* using the username *me* and the password *mypassword*.

```
<vsslabel vsspath="/source/aProject/myfile.txt"
          version="4"
          label="1.03.004"/>
```

Labels version 4 of the VSS file *$/source/aproject/myfile.txt* with the label *1.03.004*. If this version already has a label, the operation (and the build) will fail.

7 IDE Integration

7.1 About Antidote

Version 0.1 (2001/02/13)

Authors: Simeon H.K. Fitch (mailto:simeon@fitch.net)

7.1.1 Overview

Antidote is the Ant (http://jakarta.apache.org/ant/index.html) subproject for developing a graphical user interface to facilitate the efficient use of Ant. In general, its purpose is to allow the quick generation, modification, and use of Ant build files, helping the user define a build process and track down build problems. It is not meant to be an IDE, but an enabler for the powerful features available in Ant, particularlyl for novice users, or users who want a rapid way of controlling their build process.

7.1.2 Status

Antidote is still in the early stages of development, but does have a set of usable features, including:

- Reading Ant build files.
- Selecting targets and executing them.
- Context highlighted build status console.
- Modification of (some) build file components.
- Saving modified build file.

Current development tasks include:

- A more complete set of target and task editing capabilities.
- A wizard for creating basic build files, including importing existing code bases.
- Better build progress monitoring.

The Antidote source distribution comes with requirements and design documentation that better cover the details of application architecture, how to develop against it, and what the long term goals are. Furthermore, there is a `TODO` file listing the detailed, near-term tasks that need accomplishing.

7.1.3 Getting Involved

The source code for Antidote is included with the CVS (http://jakarta.apache.org/site/cvsindex.html) version of Ant (http://jakarta.apache.org/cvsweb/index.cgi/jakarta-ant), starting in the directory jakarta-ant/src/antidote (http://jakarta.apache.org/cvsweb/index.cgi/jakarta-ant/src/antidote). All the existing documentation can be found there where new contributors should read:

- Design Overview (http://jakarta.apache.org/cvsweb/index.cgi/~checkout~/jakarta-ant/src/antidote/docs/design-overview.html)
- Feature List (http://jakarta.apache.org/cvsweb/index.cgi/~checkout~/jakarta-ant/src/antidote/docs/gui-requirements.html)
- Idea Refinement (http://jakarta.apache.org/cvsweb/index.cgi/~checkout~/jakarta-ant/src/antidote/docs/gui-ideas.txt)
- New Module HOWTO (http://jakarta.apache.org/cvsweb/index.cgi/~checkout~/jakarta-ant/src/antidote/docs/new-module-howto.html)
- Static Class Diagrams (http://jakarta.apache.org/cvsweb/index.cgi/~checkout~/jakarta-ant/src/antidote/docs/uml/index.html)

Online discussions about Antidote occur on the jakarta-ant mailing list (http://jakarta.apache.org/site/mail.html). The application infrastructure is fairly complete, but there are almost unlimited oppotunities for feature contributions.

Asipring contributors new to the Jakarta Project should (carefully) read the following for details on the contribution process:

- Get Involved (http://jakarta.apache.org/site/getinvolved.html)
- Project Guidelines (http://jakarta.apache.org/site/guidelines.html)
- Source Repositories (how to contribute patches) (http://jakarta.apache.org/site/source.html)

7.2 AntRunner For JBuilder (unbundled)

See http://www.dieter-bogdoll.de/java/AntRunner/.

7.3 NetBeans Forte for Java (unbundled)

See http://ant.netbeans.org/.

7.4 Visual Age for Java Tasks and Plugin User Manual

by

- Wolf Siberski (wolf.siberski@tui.de)
- Christoph Wilhelms (christoph.wilhelms@tui.de)

Version 1.1 - 2001/02/14

7.4.1 Table of Contents

- Introduction [p 185]
- The Tasks [p 186] [p 185]
  - VAJLoad [p 186]
  - VAJExport [p 187]
  - VAJImport [p 188]
- A sample build file [p 188]
- The Plugin [p 190]
  - Installation [p 190]
  - Usage [p 191]
- Frequently Asked Questions [p 192]
- Visual Age Versions [p 193]
- History [p 194] [p 194]

7.4.2 Introduction

Visual Age for Java is a great Java IDE, but it lacks decent build support; for creating deliveries. On the other hand, Ant supports the build process very good, but is (at least at the moment) command line based. So we decided to write some tasks to access the VAJ repository and a small visual Ant frontend to make running Ant from VAJ possible. We use the Tool API to integrate Ant in VisualAge for Java. In combination with the VAJ tasks (vajload, vajexport, vajimport) you can load defined defined versions of projects

into your workspace, export the source code, compile it with an external compiler and build a jar without leaving the IDE. Of course compile messages are viewed in a logging window. Concluding: This tool provides decent deployment support VAJ has not (out of the box).

7.4.3 The Tasks

At the moment there are three tasks which help integrating the VAJ repository contents into an external build process:

VAJLoad [p 186]	loads specified versions into the workspace
VAJExport [p 187]	exports specified packages into the file system
VAJImport [p 188]	imports specified files into the workspace

These tasks are described in detail below.

7.4.4 VAJLoad

7.4.4.1 Description:

Loads a specified VAJ project version into the workspace.

7.4.4.2 Parameters

none

7.4.4.3 Parameters specified as nested elements

7.4.4.3.1 vajproject

Attribute	Description	Required
name	name of the VAJ project to load into the workspace	yes
version	name of the requested version	yes

7.4.4.4 Example

```
<vajload>
    <vajproject name="My Testcases" version="1.7beta" />
    <vajproject name="JUnit" version="3.2" />
</vajload>
```

7.4.5 VAJExport

7.4.5.1 Description:

Exports Java source files, class files and/or resources from the workspace to the file system. Exports can be specified by giving the VAJ project name and package name(s). This works very similar to FileSets [p 20] .

7.4.5.2 Parameters

Attribute	Description	Required
destdir	location to store the exported files	yes
exportSources	export source files (default: "yes")	no
exportResources	export resource files (default: "yes")	no
exportClasses	export class files (default: "no")	no
exportDebugInfo	include debug info in exported class files (default: "no")	no
defaultexcludes	use default excludes when exporting (default: "yes")	no

7.4.5.3 Parameters specified as nested elements

7.4.5.3.1 include

specifies the packages to include into the export

Attribute	Description	Required
name	name of the VAJ project and package to export. The first element of the name must be the project name, then the package name elements separated by '/'.	yes

7.4.5.3.2 exclude

specifies the packages to exclude from the export

Attribute	Description	Required
name	name of the VAJ project/package not to export	yes

7.4.5.4 Example

```
<vajexport destdir="${src.dir}" exportResources="no">
    <include name="MyProject/**"/>
    <exclude name="MyProject/test/**"/>
</vajexport>
```

This example exports all packages in the VAJ project 'MyProject', except packages starting with 'test'.

7.4.6 VAJImport

7.4.6.1 Description:

Imports Java source files, class files and/or resources from the file system into VAJ. These imports can be specified with a fileset.

7.4.6.2 Parameters

Attribute	Description	Required
vajProject	imported files are added to this VAJ project	yes
importSources	export source files (default: "yes")	no
importResources	export resource files (default: "yes")	no
importClasses	export class files (default: "no")	no

7.4.6.3 Parameters specified as nested elements

7.4.6.3.1 fileset

A FileSet [p 20] specifies the files to import.

7.4.6.4 Example

```
<vajimport project="Test" importClasses="true">
    <fileset dir="${import.dir}">
        <include name="com/sample/**/*.class"/>
        <exclude name="com/sample/test/**"/gt;
    </fileset>
</vajimport>
```

This example imports all class files in the directory ${import.dir}/com/sample excluding those in the subdirectory test

---

7.4.7 A sample build file

This is a sample build file which builds a zip file useful for distribution of the VAJ plugin. The build file exports a specific version of Ant, compiles it, extracts necessary helper classes and zips everything together. This is useful as complete example as well as to deploy Ant in a team where all developers use VAJ. The resulting zip file must be unzipped in `<VAJInstallDir>\ide\tools\org-apache-tools-ant`. For more information see the installation [p 190] section.

```
<?xml version="1.0"?>
<!-- ===================================================================== -->
<!-- Builds a binary distribution of the VAJ Ant Plugin                    -->
<!-- ===================================================================== -->
<project name="anttool" default="buildall" basedir="c:\temp\anttool">

  <property name="src.dir" value="${basedir}/src"/>
  <property name="zip.dir" value="${basedir}/dist"/>
  <property name="build.classes" value="${basedir}/classes"/>
```

```
<!-- location of the unzipped Ant source distribution  -->
<property name="antdistribution.dir" value="u:\ant-1.3beta1"/>

<!-- VAJ install dir -->
<property name="vaj.dir" value="C:\IBMVJava2"/>

<path id="classpath">
    <pathelement location="${src.dir}"/>
    <pathelement location="${vaj.dir}\IDE\project_resources\IBM IDE Utility class libraries" />
    <pathelement location="${build.classes}"/>
</path>


<!-- ================================================================== -->
<!-- Load Projects into Workspace                                       -->
<!-- ================================================================== -->
<target name="load" description="load projects">
  <vajload>
    <project name="Ant" version="1.3"/>
    <project name="Apache Oro RegExp" version="2.0.1"/>
  </vajload>
</target>


<!-- ================================================================== -->
<!-- Creates the build dirs                                             -->
<!-- ================================================================== -->
<target name="preparedirs">
  <mkdir dir="${basedir}"/>
  <mkdir dir="${src.dir}"/>
  <mkdir dir="${zip.dir}"/>
  <mkdir dir="${build.classes}"/>
</target>


<!-- ================================================================== -->
<!-- Export from VA                                                     -->
<!-- ================================================================== -->
<target name="export" depends="preparedirs" description="export projects">
  <mkdir dir="${src.dir}"/>
  <!-- Export all Ant sources except optional tasks and the Oro sources -->
  <vajexport destdir="${src.dir}">
    <include name="Ant/**"/>
    <exclude name="Ant/**/optional/**"/>
    <exclude name="Ant/org/apache/tools/ant/gui/**"/>
    <include name="Apache Oro*/**"/>
  </vajexport>
  <!-- Export selected optional tasks -->
  <vajexport destdir="${src.dir}">
    <include name="Ant/org/apache/tools/ant/taskdefs/optional/ide/**"/>
  </vajexport>
</target>

<!-- ================================================================== -->
<!-- Unzip necessary libraries                                          -->
<!-- ================================================================== -->
<target name="expand">
  <mkdir dir="${build.classes}"/>
  <unzip src="${antdistribution.dir}\lib\parser.jar"
         dest="${build.classes}"
  />
  <unzip src="${antdistribution.dir}\lib\jaxp.jar"
         dest="${build.classes}"
  />
  <delete dir="${build.classes}\META-INF" />
</target>

<!-- ================================================================== -->
<!-- Compile the source code                                            -->
<!-- ================================================================== -->
<target name="compile" depends="expand" description="compile java sources">
  <mkdir dir="${build.classes}"/>
  <javac srcdir="${src.dir}"
         destdir="${build.classes}"
         debug="on"
         deprecation="off"
         optimize="on" >
    <classpath refid="classpath" />
    <include name="org/apache/**"/>
    <exclude name="**/JakartaRegexpMatcher.java"/>
  </javac>
</target>

<!-- ================================================================== -->
<!-- Copy resources and zip everything together                         -->
<!-- ================================================================== -->
<target name="zip" depends="compile" description="creates zip distribution">
  <copy todir="${build.classes}">
    <fileset dir="${src.dir}">
        <include name="org/apache/**"/>
        <exclude name="**/*.java"/>
    </fileset>
  </copy>
  <mkdir dir="${build.classes}/doc"/>
  <copy todir="${build.classes}/doc" overwrite="yes">
```

```
      <fileset dir="${antdistribution.dir}/docs">
          <include name="VAJAnttool.html, toolmenu.gif, anttool1.gif"/>
      </fileset>
    </copy>
    <copy file="${antdistribution.dir}/src/main/org/apache/tools/ant/taskdefs/optional/ide/default.ini" todir="${build.classes}" overwrite="yes"/>
    <mkdir dir="${zip.dir}"/>
    <zip basedir="${build.classes}" zipfile="${zip.dir}/anttool.zip" />
  </target>

  <!-- ================================================================== -->
  <!-- Cleans source and class dirs                                       -->
  <!-- ================================================================== -->
  <target name="clean" depends="preparedirs" description="removes all files from src and build tree">
    <delete>
      <fileset dir="${src.dir}"/>
    </delete>
    <delete>
      <fileset dir="${build.classes}"/>
    </delete>
    <delete>
      <fileset dir="${zip.dir}"/>
    </delete>
  </target>

  <!-- ================================================================== -->
  <!-- Cleans the build dir, loads required project versions, exports,    -->
  <!-- compiles and zips the Plugin                                       -->
  <!-- ================================================================== -->
  <target name="buildall" depends="clean, load, export, zip" description="build all">
  </target>
</project>
```

7.4.8 The Plugin

The tasks are usable within VAJ by running the org.apache.tools.ant.Main class, but this is quite inconvenient. Therefore a small GUI is provided which allows selecting a build file and executing its targets. This Plugin is accessible from the VAJ Tools menu (see Usage [p 191]).

7.4.9 Installation

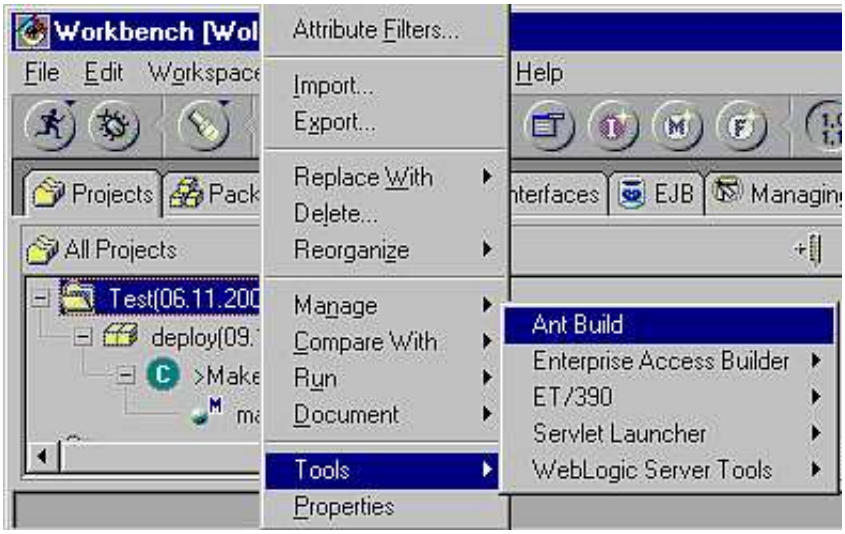
At the moment the installation has it's rough edges. If something described below doesn't work for You, it's probably not Your fault but incomplete/wrong instructions. In this case, please contact one of the authors [p 185] . We assume C:\IBMVJava as VAJ install directory. If You have installed it elsewhere, adapt the pathes below. Execute following steps to get the PlugIn up and running:

- install the Visual Age IDE Tools (via File->Quick Start-> Add feature->'IBM IDE Utility class libraries'
- import an appropriate XML parser to VAJ (we use Xerces 1.2.0 and are happy with it). Unfortunately the XML parser delivered with VAJ (in the project 'IBM XML Parser for Java') doesn't work with Ant. You have to remove that project (temporarily) from the workspace before importing another XML implementation.
- import the Ant sources into VAJ.
- Create the directory C:\IBMVJava\ide\tools\org-apache-tools-ant.
- export the Ant and XML parser class files into this directory. Be sure to select class files and resources. Sources don't have to be exported. Some optional tasks have errors and can't be exported when You don't have the necessary packages in Your workspace (e.g. junit task, ejbc task). If You need this tasks either import these packages into VAJ, too, or copy the .class files directly from the binary distribution.
- copy default.ini (in jakarta-ant\src\...\taskdefs\optional\ide) to C:\IBMV-Java\ide\tools\org-apache-tools-ant\default.ini.

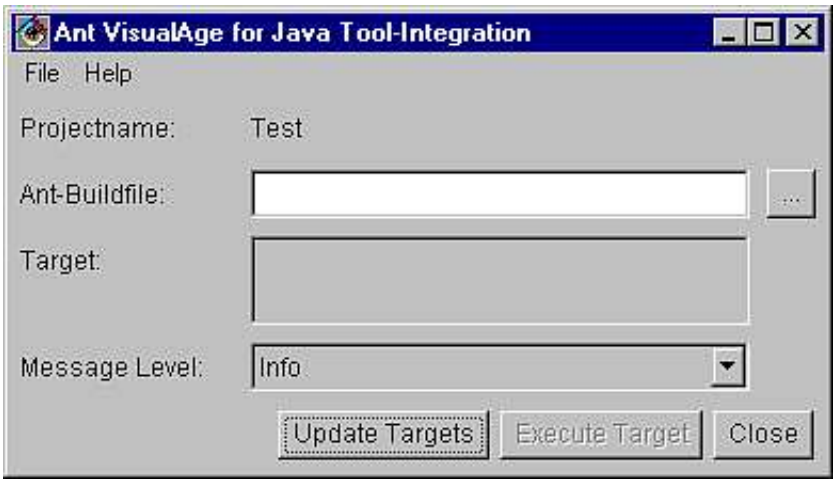
- if you want to access this help from the Workbench, create the directory C:\IBMV-Java\ide\tools\org-apache-tools-ant\doc and copy the files VAJAntTool.html, toolmenu.gif and anttool1.gif to it.
- VAJ has to be restarted to recognize the new tool.
- Now if You open the context menu of a project, You should see the entry 'Ant Build' in the Tools submenu (see Usage [p 191]).
- Make sure the tool works as expected. Now You can remove Ant and the imported XML parser from Your workspace (and optionally add the IBM parser again).

7.4.10 Usage

Beeing sure the tool is installed correctly and your Ant build file is configured, it is really easy to use.
Go to your Workbench, select the project you want to deploy and open its context menu. In the submenu *Tools* you should find the new entry *Ant Build*. Klick it to start the tool!



After a short time this frame should pop up:



This frame contains the following elements:

- A menubar with some options described later
- The name of your selected VAJ project
- An entry field for the Ant XML buildfile with a browse [...] button. The full qualified filename, including the directory is needed here.
- A list with tasks specified in the buildfile. Until your first save of the build info (described later), this list will be empty. When loading a build file by the *Reload Project* button, this list is filled with all tasks which have a description attribute. The task you select in this list will be executed when pressing the *Execute Task* button.
- A pulldown box for specifying the log level.
- Three buttons. Two of them I have already described and the third one is just the *Close* button to exit our small tool!

After you have set up your buildprocess you might find it useful to save the data you've just entered, so we implemented an option to save it to the repository into your selected project. Make sure that you have an open edition of your project before selecting *Save BuildInfo To Repository* from the *File* menu. Now your information is saved to this edition of your project and will be loaded automatically the next time you start *Ant Build*.
If you have closed the log window accidentally, it can be reopened with the *Log* item in the *File* menu, and if you want to know who developed this, just select *About* in the *Help* menu.

7.4.11 Frequently Asked Questions

Q: I want to load, export and build more then one Visual Age project to one jar! How to?
A: The VA tasks are able to load and export several Projects all at once. You can choose whatever project you like for storing the tool information, it doesn't really matter

Q: When I load my build file, the list of targets is empty. Why?
A: You need to add the optional "description" parameter to the targets You want to come up in the list. Then reload the build file in the "ant build" tool. We chose to display only targets with description to allow

the build file developer to distinguish between targets for end users and helper targets.

Q: Is there a sample build file available?
A: Now You can find an example [p 188] in this manual

Q: Why does it export my entire workspace when I've already implicitly selected a project when starting the Tool?
A: This selection does not carry into the buildfile you are using. Set the Project name at the beginning of the "includes" parameter.

Q: When I import Ant into my Workspace, I get Problems reported. Can I ignore them?
A: It depends on the problems reported, and what you want to do with Ant. Problems you can't ignore:

- Classes from javax.xml.parser missing - install a compatible parser (see installation [p 190])
- Classes from com.ibm.ivj.util missing - install the Visual Age IDE Utility feature (see installation [p 190]).

Q: Why is the task name vajload when the class is called VAJLoadProjects?
A: At the moment this task can load only project versions. This is reflected by the class name. It may be extended to load package and class versions, too, and we didn't want to let these extensions affect the build files. Therefore we chose the more general name vajload as task name.

Q: I want to use the same buildfile both within Visual Age and from the command line using my regular Ant environment. What do I need to be aware of?
A: The three Visual Age Tasks won't work when executing Ant from the command line.

Q: How do I control the import/export of sourcefiles, compiled files and project resources explicity?
A: Via the Boolean values exportClasses (default false) exportSources (default true) and exportResources (default true). In some situations, Resources are not exported correctly without this being explicity set.

7.4.12 Known Problems

- Exporting a package containing just resources doesn't work. This is a VAJ Tool API bug. Workaround: create a dummy class and set 'exportSources' to false.

7.4.13 VisualAge for Java Versions

This tool integration has been tested with versions 3.02 and 3.5 of VisualAge for Java. It should run with the 2.x Versions, too, but we didn't try. The graphical user interface is build with AWT so it is JDK independent by now.

7.4.14 History

1.0	2000/09/11	Initial Version
1.1	2001/02/14	Added Task documentation and more FAQs (thanks to Richard Bourke for the FAQ additions)

8 Developing with Ant

8.1 Writing Your Own Task

It is very easy to write your own task:

1. Create a Java class that extends `org.apache.tools.ant.Task`.
2. For each attribute, write a *setter* method. The setter method must be a `public void` method that takes a single argument. The name of the method must begin with `set`, followed by the attribute name, with the first character of the name in uppercase, and the rest in lowercase. The type of the attribute can be:
   - `String`
   - any primitive type (they are converted for you from their String-representation in the buildfile)
   - boolean – your method will be passed the value *true* if the value specified in the buildfile is one of `true`, `yes`, or `on`)
   - `Class`
   - `File` (in which case the value of the attribute is interpreted relative to the project's basedir)
   - any other type that has a constructor with a single `String` argument
3. If your task has enumerated attributes, you should consider using a subclass of `org.apache.tools.ant.types.EnumeratedAttribute` as an argument to your setter method. See `org/apache/tools/ant/taskdefs/FixCRLF.java` for an example.
4. If the task should support character data, write a `public void addText(String)` method.
5. For each nested element, write a *create* or *add* method. A create method must be a `public` method that takes no arguments and returns an `Object` type. The name of the create method must begin with `create`, followed by the element name. An add method must be a `public void` method that takes a single argument of an `Object` type with a no-argument constructor. The name of the add method must begin with `add`, followed by the element name.
6. Write a `public void execute` method, with no arguments, that throws a `BuildException`. This method implements the task itself.

8.1.1 The Life-cycle of a Task

1. The task gets instantiated using a no-argument constructor, at parser time. This means even tasks that are never executed get instantiated.
2. The task gets references to its project and location inside the buildfile via its inherited `project` and `location` variables.
3. If the user specified an `id` attribute to this task, the project registers a reference to this newly created task, at parser time.
4. The task gets a reference to the target it belongs to via its inherited `target` variable.
5. `init()` is called at parser time.
6. All child elements of the XML element corresponding to this task are created via this task's `createXXX()` methods or instantiated and added to this task via its `addXXX()` methods, at parser time.
7. All attributes of this task get set via their corresponding `setXXX` methods, at runtime.
8. The content character data sections inside the XML element corresponding to this task is added to the

task via its `addText` method, at runtime.

9. All attributes of all child elements get set via their corresponding `setXXX` methods, at runtime.
10. `execute()` is called at runtime. While the above initialization steps only occur once, the execute() method may be called more than once, if the task is invoked more than once. For example, if `target1` and `target2` both depend on `target3`, then running `'ant target1 target2'` will run all tasks in `target3` twice.

8.1.2 Example

Let's write our own task, which prints a message on the `System.out` stream. The task has one attribute, called `message`.

```
package com.mydomain;

import org.apache.tools.ant.BuildException;
import org.apache.tools.ant.Task;

public class MyVeryOwnTask extends Task {
  private String msg;

  // The method executing the task
  public void execute() throws BuildException {
    System.out.println(msg);
  }

  // The setter for the "message" attribute
  public void setMessage(String msg) {
    this.msg = msg;
  }
}
```

It's really this simple ;-)

Adding your task to the system is rather simple too:

1. Make sure the class that implements your task is in the classpath when starting Ant.
2. Add a `<taskdef>` element to your project. This actually adds your task to the system.
3. Use your task in the rest of the buildfile.

8.1.3 Example

```
<?xml version="1.0"?>

<project name="OwnTaskExample" default="main" basedir=".">
  <taskdef name="mytask" classname="com.mydomain.MyVeryOwnTask"/>

  <target name="main">
    <mytask message="Hello World! MyVeryOwnTask works!"/>
  </target>
</project>
```

Another way to add a task (more permanently), is to add the task name and implementing class name to the `default.properties` file in the `org.apache.tools.ant.taskdefs` package. Then you can use it as if it were a built-in task.

8.2 Build Events

Ant is capable of generating build events as it performs the tasks necessary to build a project. Listeners can be attached to Ant to receive these events. This capability could be used, for example, to connect Ant to a GUI or to integrate Ant with an IDE.

To use build events you need to create an ant `Project` object. You can then call the `addBuildListener` method to add your listener to the project. Your listener must implement the `org.apache.tools.antBuildListener` interface. The listener will receive BuildEvents for the following events

- Build started
- Build finished
- Target started
- Target finished
- Task started
- Task finished
- Message logged

If you wish to attach a listener from the command line you may use the `-listener` option. For example:

```
ant -listener org.apache.tools.ant.XmlLogger
```

will run Ant with a listener that generates an XML representation of the build progress. This listener is included with Ant, as is the default listener, which generates the logging to standard output.

9 Ant Resources

9.1 FAQ

There is an on-line FAQ for Ant at jakarta.apache.org (http://jakarta.apache.org/jyve-faq/Turbine/screen/DisplayTopics/action/SetAll/project_id/2/faq_id/16). This FAQ is interactive, which means you can ask and answer questions on line.

One of the questions popping up quite often is "Is there a DTD for buildfiles?" Please refer to the FAQ for an answer.

10 License


```
/*
 * ====================================================================
 *                    The Apache Software License, Version 1.1
 * ====================================================================
 *
 *    Copyright (C) 1999 The Apache Software Foundation. All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without modifica-
 * tion, are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of  source code must  retain the above copyright  notice,
 *    this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright notice,
 *    this list of conditions and the following disclaimer in the documentation
 *    and/or other materials provided with the distribution.
 *
 * 3. The end-user documentation included with the redistribution, if any, must
 *    include  the following  acknowledgment:  "This product includes  software
 *    developed  by the  Apache Software Foundation  (http://www.apache.org/)."
 *    Alternately, this  acknowledgment may  appear in the software itself,  if
 *    and wherever such third-party acknowledgments normally appear.
 *
 * 4. The names "Ant" and  "Apache Software Foundation"  must not be used to
 *    endorse  or promote  products derived  from this  software without  prior
 *    written permission. For written permission, please contact
 *    apache@apache.org.
 *
 * 5. Products  derived from this software may not  be called "Apache", nor may
 *    "Apache" appear  in their name,  without prior written permission  of the
 *    Apache Software Foundation.
 *
 * THIS SOFTWARE IS PROVIDED ‘‘AS IS’’ AND ANY EXPRESSED OR IMPLIED WARRANTIES,
 * INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
 * FITNESS  FOR A PARTICULAR  PURPOSE ARE  DISCLAIMED.  IN NO  EVENT SHALL  THE
 * APACHE SOFTWARE  FOUNDATION  OR ITS CONTRIBUTORS  BE LIABLE FOR  ANY DIRECT,
 * INDIRECT, INCIDENTAL, SPECIAL,  EXEMPLARY, OR CONSEQUENTIAL  DAMAGES (INCLU-
 * DING, BUT NOT LIMITED TO, PROCUREMENT  OF SUBSTITUTE GOODS OR SERVICES; LOSS
 * OF USE, DATA, OR  PROFITS; OR BUSINESS  INTERRUPTION)  HOWEVER CAUSED AND ON
 * ANY  THEORY OF LIABILITY,  WHETHER  IN CONTRACT,  STRICT LIABILITY,  OR TORT
 * (INCLUDING  NEGLIGENCE OR  OTHERWISE) ARISING IN  ANY WAY OUT OF THE  USE OF
 * THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 * This software  consists of voluntary contributions made  by many individuals
 * on behalf of the  Apache Software Foundation.  For more  information  on the
 * Apache Software Foundation, please see <http://www.apache.org/>.
 *
 */
```

11 Feedback

To provide feedback on this software, please subscribe to the Ant User Mailing List (http://jakarta.apache.org/site/mail.html)

If you want to contribute to Ant or stay current with the latest development, join the Ant Development Mailing List (http://jakarta.apache.org/site/mail.html)

Archives of both lists can be found at http://archive.covalent.net/. Many thanks to Covalent Technologies. A searchable archive can be found at http://marc.theaimsgroup.com. If you know of any additional archive sites, please report them to the lists.